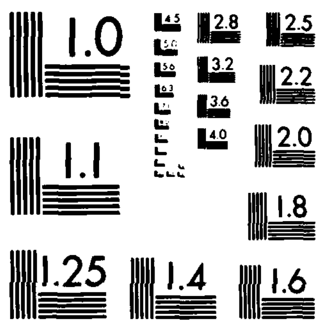
AD-A151 833 MICROCOMPUTER APPLICATION OF AEROSPACE ASSET SURFACE SEARCH PLANNING(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING 1/2

UNCLASSIFIED D R DOUGLAS 14 DEC 84 AFIT/GSO/MATH/84D-1 F/G 6/7 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

MICROCOMPUTER APPLICATION OF AEROSPACE ASSET SURFACE SEARCH PLANNING

THESIS

Don Richard Douglas
Captain, USAF

AFIT/GSO/MATH/84D-1

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GSO/MATH/84D-1

MICROCOMPUTER APPLICATION OF AEROSPACE ASSET SURFACE SEARCH PLANNING

THESIS

Don Richard Douglas
Captain, USAF

AFIT/GSO/MATH/84D-1

Approved for public release; distribution unlimited

AFIT/GSO/MATH/84D-1

MICROCOMPUTER APPLICATION OF
AEROSPACE ASSET SURFACE SEARCH PLANNING

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of

Requirements for the Degree of

Master of Science in Space Operations

Don Richard Douglas

Captain, USAF

December 1984

Approved for public release; distribution unlimited

Acknowledgments

I wish to thank my thesis advisor, Dr. Panna B. Nagarsenker, Professor of Mathematics at the Air Force Institute of Technology (AFIT), for her guidance and support throughout this thesis effort. Our regularly-scheduled meetings provided me with ingenious insights into program design and kept me firmly goal-oriented. The assistance and recommendations of my reader, Lt. Colonel Joseph W. Coleman, USAF, Professor of Operations Research, were also of great merit.

Additionally, I am indebted to the faculty of the U. S. Coast Guard's (USCG) National Search and Rescue (SAR) School in New York City, especially Major Charles L. Vessey, USAF. He provided his SAR expertise/insight, recommendations to enhance the usefulness of the designed programs, and sample problems with solutions, enabling validation of the search programs generated for this project.

Equally noteworthy was the extensive computer assistance of LTJG Dale G. Streyle, USCG Atlantic Area Command (New York). He helped transfer, compile, link, and get the attached software running on the Coast Guard's standard mainframe computer system (C-3). This enabled the compiled/linked programs to be immediately used by National SAR School faculty, as well as allowing their +65 pages of source code to be electronically downloaded to interested SAR students having the microcomputers to run them on.

Finally, special appreciation and love goes to the former Paula J. Hartzfeld of Bronaugh, Missouri. After foolishly marrying for love instead of money, she valiantly picked up my family responsibilities, in addition to her own, releasing me for full-time AFIT study and thesis research. I also acknowledge the unique contributions of my son, Craig, who ensured never a dull moment in our household during my course of study.

Don Richard "Rick" Douglas

Contents

Appendix B: Search Planning Software User's Guide

Appendix C: Sample Problem, Program(s) Run & Solutions

Appendix D: Search Planning Program(s) Listing (Source Code)

List Of Figures

Abstract

This paper tackles the most time-consuming and complicated type of search and recovery planning -- calculating the approximate surface position of an aerospace object which has been affected over time by glide or parachute winds aloft, as well as surface current winds, leeway drift, and sea current vectors. The three, highly-interactive, search applications programs herein are written in Standard Pascal using Borland International's "TURBO Pascal" (an inexpensive software package available for virtually every microcomputer on the market). They have been tested on a small, portable, 64K memory, Z-80A processor-based microcomputer (Osborne One), a Convergent Technologies C-3 Data System, and a Digital Equipment Corporation VAX 11-780 mainframe.

Search and Rescue/Recovery (SAR) in the United States is based on the humanitarian principle which compels people to render aid to those in distress. Search planning guidelines and formulae to help locate persons in distress or missing aerospace objects are described in the National Search and Rescue Manual (AFM 64-2). This methodology has not been implemented for microcomputers in a compiled, transportable programming language like Pascal. This research project does just that. It does not, however,

teach the guidelines or formulae. The reader MUST have a solid understanding of SAR methodology before using the attached software package to assist in making decisions where human life is at risk. In fact, since no amount of testing can uncover 100% of program errors, the attached software package is recommended for training use only.

Appendices include: glossary, user's guide, sample problem with program runs and solutions, and source code listing.

Chapter 1

MICROCOMPUTER APPLICATION OF AEROSPACE ASSET SURFACE SEARCH PLANNING

Background

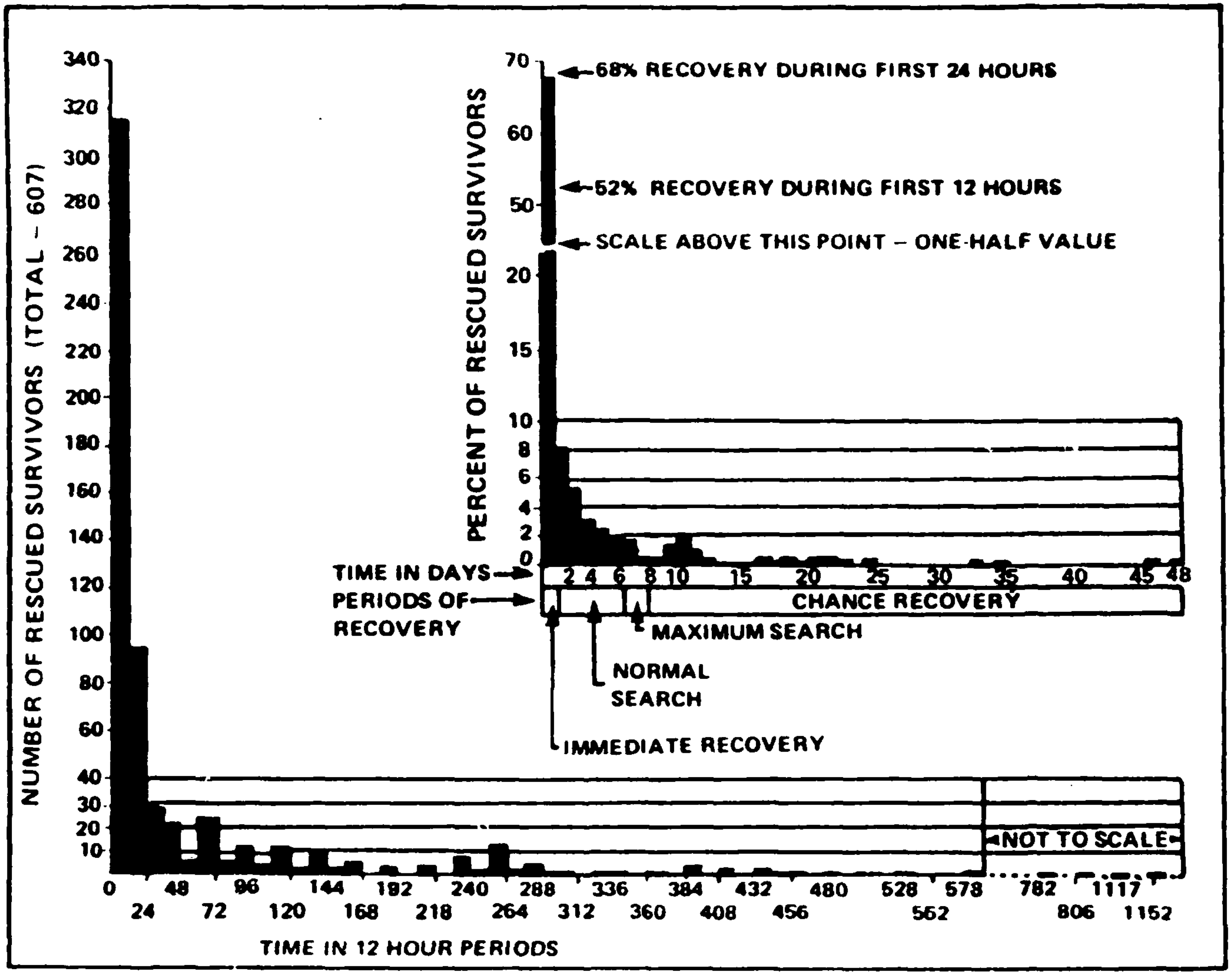
What is the value of a human life? Each year millions of dollars and countless manhours are expended searching for missing persons in distress. International, national, state, and local government search and rescue/recovery (SAR) agencies commit limited resources looking for the real or perceived victims of accidents or incidents such as sinking vessels or airliner or light plane crashes. And, as humans increasingly orbit, fly and sail over the Earth, the number of such missing/distressed persons can only multiply.

Likewise, as man launches and recovers increasing amounts of equipment to and from space, more of it will, undoubtedly, land somewhere other than intended. Remember when no one knew where the uncontrolled US Skylab or the Soviet nuclear-powered COSMOS satellite(s) would crash once they re-entered the Earth's atmosphere?:

> There is no way to tell exactly where between 50 North and 50 South latitude Skylab debris will fall even as late as one orbit prior to entry. . . Skylab weighs about 175,000 lbs., and analysis indicates that 40,000-50,000 lbs. could survive reentry. (5:19)

No one knew where these spacecraft finally did land until several hours/days of searching located the missing pieces (1:33,7:1-6,9:1-5).

Survivor Recovery Time

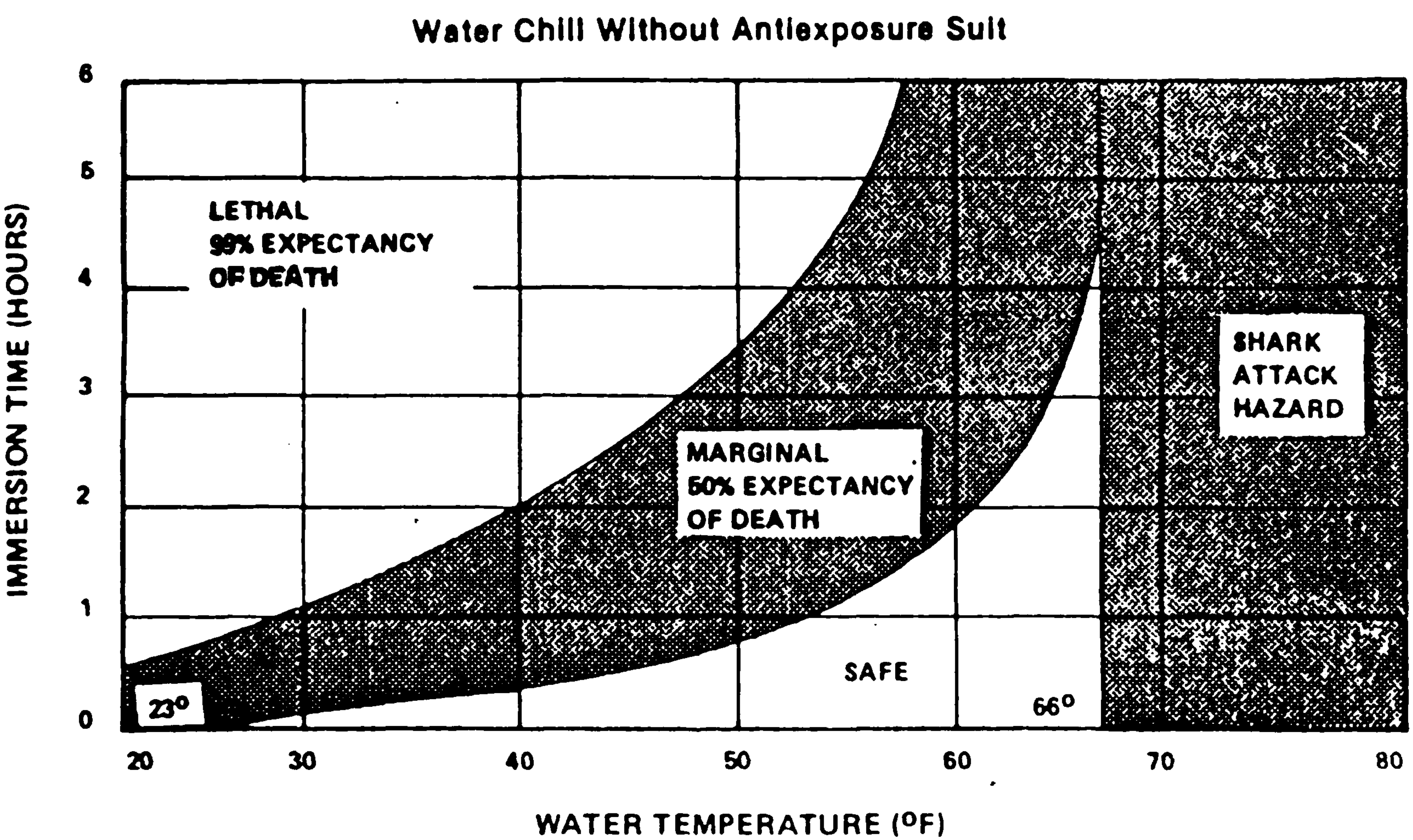


The United States Coast Guard (USCG) is tasked with training Federal military and civil agencies in search planning (10:2-7). For this purpose they maintain the National SAR School in New York City. After tabulating many years of search statistics, USCG developed a lengthy, yet comprehensive, manual method of search area determination using vector addition, algebra, tables, graphs, and nomographs. This technique is taught over a 3-week course, where the typical oceanic problem takes experienced planners, assisted by electronic calculators, several hours to complete (8).

Unfortunately, these same hours may be spent by injured survivors on land or in the water awaiting recovery (see Figure 1). Or, valuable equipment may receive environmental exposure damage (e.g., corrosive seawater) before search forces are directed where to look for it.

Problem Description

The National SAR School manual search planning technique is not available for microcomputer users in a widely-transferable, high-level, compiled programming language, like Pascal (8). Such a software package could save precious time (see Figure 2) by providing appropriate agencies with preliminary data on which to initiate search planning. This would result in search forces being organized and launched much sooner, saving more lives and critical resources. The search planning technique is


Water Chill Without Antiexposure Suit
LETHAL 99% EXPECTANCY OF DEATH
MARGINAL 50% EXPECTANCY OF DEATH
SHARK ATTACK HAZARD
SAFE
23°
66°
IMMERSION TIME (HOURS)
0
1
2
3
4
5
6
WATER TEMPERATURE (°F)
20
30
40
50
60
70
80

available, however, on the USCG mainframe computer system at Governor's Island, New York, with some limitations on capability and accessibility.

Any discussion of these limitations must first describe current capabilities. An operations research firm in Paoli, PA, received a government contract to design a "Computer Assisted Search Planning" (CASP) program to run on the Coast Guard's Convergent Technologies C-3 Data System computer. The program allows weighted multi-criteria decision-making, draws search location probability maps, and has other features which only embellish the basic manual calculation method and would never fit in the limited memory space of the standard 48-64K RAM microcomputer. Yet, CASP only handles surface search object planning! It lacks the capability of calculating the initial surface/splash-down position of gliding, falling, and/or parachuting aerospace objects (ejecting pilots, satellite packages, etc.). For years, the National SAR School faculty has been forced to manually calculate this aerospace drift position before they could execute their commercially-procured, mainframe computer search planning software (8). Were this aerospace drift capability incorporated into the software created in this thesis effort, then it could be used on the USCG mainframe and microcomputers alike.

The second limitation involves accessibility. Outside agencies desiring rapid and accurate search planning

assistance do not have direct access to USCG's mainframe running the CASP program. They must verbally pass the required information by phone to the Coast Guard, who must key in their data and phone back the results. This method is slow and typographical errors can make it an even longer process. Adding to the problem are possible time-sharing delays, explained by Fraley and Kem:

> Due to the increased use of computers in all aspects of management, the number of users attempting to access the computer is normally quite large. This aspect may then cause the response time of non-dedicated remote computer systems to be unacceptable in a time critical environment. (2:4)

These authors conclude with the valid suggestion that microcomputers offer a solution to this accessibility problem.

As a final advantage of developing search planning software for microcomputers, the Coast Guard has expressed interest in freely offering/downloading this software to interested students/graduates on an "optional basis" to maintain the skills they learned at the National SAR School (8).

Research Objective

The primary objective of this project is to design, implement, verify and validate a microcomputer-based aerospace asset search planning tool. The reason is to generate answers much faster, and with greater accuracy, than if manual calculations were used.

The other key objective is to create a search planning tool that is highly-transportable between different types of microcomputers. This entails sub-objectives of keeping the program(s) small and writing it (them) in a high-level, transportable programming language for microcomputers. Therefore, the tool is divided into three programs. This keeps their compiled versions small enough to implement on microcomputers having at least 64K random-access memory (RAM) and one disk drive. The selection of programming language is now described.

Programming Language Selection

The choice of a suitable microcomputer language offered these alternatives: BASIC, FORTRAN, or Pascal. Most microcomputers can be purchased today with their own generic brand of BASIC (Beginner's All-Purpose Symbolic Instruction Code) language. Unfortunately, most microcomputer BASICs are slower, interpreted or pseudo-compiled languages. Interpreted languages translate each program statement into a sequence of machine code instructions which are executed before the next statement is translated. This method takes much longer to run than does pre-translating the program into machine code (or "compiling it", as do Fortran and Pascal) enabling instant execution. Also, BASIC does not allow the use of mnemonic names to give the user a clue as to a variable's purpose in a program. Worst of all, BASICs are often incompatible between computer

types, limiting their portability.

The FORTRAN (FORmula TRANslation) story is even more bleak. Those packages available for microcomputers are non-standard subsets of FORTRAN 77. At time of writing only one software company (Digital Research) had announced development of a complete, standard version of FORTRAN 77 for micros. However, this version only runs on IBM personal computers upgraded (at user expense) to 512K RAM and is scheduled to cost over ten times as much as the language used in this project.

By this process of elimination, Pascal was ultimately selected. This language was designed by Professor Niklaus Wirth of the Eidgenossische Technische Hochschule in Zurich, Switzerland, and named in honor of Blaise Pascal (1623-1662), the famous 17th-century French philosopher and mathematician (6:2-3). Since first introduced in 1971, Pascal has been applied to almost any task on computers and has received increasing attention and use in business and academia worldwide. A recent Soviet journal reports:

> Scientists of the Lithuanian Academy of Sciences' Institute of Mathematics and Cybernetics, Department of Systems Programming, selected the so-called Pascal language from several programming languages used by computers (for training programmers). This language reflects the main concepts of programming rather clearly and is highly-suitable for training purposes . . . helping (programmers) to master the fundamentals of programming more quickly and easily. (11:2)

It is the only microcomputer language to meet the criteria

of being readily-available, highly-transportable, and:

> Pascal is trim enough to run in the 48K- to 64K-byte memory limit that characterizes (most personal computers). It is small enough to be easily implemented, and its trimness makes it syntax and semantics easy to specify and relatively easy to grasp. (4:234)

Best of all, Borland International's "TURBO Pascal" used in this project comes complete with a 259-page User's Manual for only $49.95 retail ($35 mail order). The attached programs were written in TURBO on an Osborne personal computer and readily compiled and executed under Standard Pascal (Jensen and Wirth rules) on a DEC VAX 11-780 mainframe.

As such, this project's three programs for aerospace asset search planning were developed to run on a minimum system consisting of at least a 64K RAM-equipped microcomputer with one disk drive. The user need not purchase TURBO to compile and execute the programs, if a pre-compiled (compiled for a specific type of personal computer) copy of these programs is acquired. The only software needed to execute the compiled programs is the user's microcomputer operating system. However, if a pre-compiled version is unavailable, the Borland product must be acquired and the source code listed in Appendix D must be keyed in and compiled to run.

Scope of Project

The scope of this project limits validation of

National SAR Manual search planning methodology, which is applied herein to microcomputer programming to calculate the smallest, feasible search area with the highest probability of locating missing aerospace objects over land or in the water. The algorithms incorporate the National SAR School's manual search planning method, making no attempts to verify the accuracy of formulae, graphs, or nomographs the Coast Guard has relied on and updated over many years of successful search and rescue experience. As such, they address: aerospace drift; calculation of average winds aloft; sea, wind, and leeway current drifts; calculation of average surface winds; minimum, maximum, and DATUM_minimax position determination; and, calculation of search area. These areas fit into the overall search planning methodology structure as shown in Figure 3.

1. Awareness
2. Initial Actions
3. Planning ———> PLANNING STAGE
4. Operations
5. Mission Conclusion

DATUM Determination

AEROSPACE INCIDENT

Calculate Aerospace Drift

Calculate Average Winds Aloft

Determine Surface Position

SURFACE INCIDENT

Determine Sea Current Drift

Determine Wind Current Drift

Determine Leeway Drift

Determine Minimum Total Drift (Dmin Position)

Determine Maximum Total Drift (Dmax Position)

Determine DATUMminimax Position

Determine Search Area

Figure 3. SEARCH STAGES

Chapter 2

Software Package Design

Overview

Software engineering practices (discussed in Chapter 3) were used throughout the development of the attached microcomputer package. Top-down design (breaking a larger problem into smaller pieces for solution and coding) led to the modular development of three programs. This research effort uses top-down program design methodology. Actual coding follows, emphasizing simplistic structures for program clarity to enable simplified modifications by individuals having limited Pascal programming experience. Then comes debugging, testing with USCG-supplied problems, and, finally, drafting of documentation (User's Guide).

Slicing Problem Into Manageable Pieces

The 64K RAM restriction set on available microcomputer memory and limited magnetic disk storage space forced the overall problem to be subdivided into smaller units of manageable size with the added advantage of faster individual modification and re-compilation times (see Figure 4). This search planning implemented in one comprehensive program would not fit most microcomputer memory or disk storage spaces. In particular, the ideal disk would have to store source (183K) and compiled (112K) code, as well as leaving approximately 10K space for input/output record

PROGRAM NAME	SOURCE CODE	COMPILED CODE	TOTAL LINES	TOTAL PAGES
AeroDrift	67K	38K	1468	27
SurfaceDrift	81K	50K	1839	34
SearchArea	35K	24K	777	15
TOTAL	183K	112K	4084	76

Figure 4. FINAL SEARCH PLANNING SOFTWARE STATISTICS

files created during program runtime. Additionally, the disk would have to reserve editing/ updating space equal to the size of the source code -- a total of 488K. For comparison, Osborne microcomputer single-sided, double-density (SSDD) disks are limited to 183K usable storage space, while single-sided, single-density formats are only 92K. The attached software fills most of two Osborne-formatted, SSDD disks, leaving approximately 65K for generated files and partial editing space.

Modular Program Development

This technique simplified coding and allowed easier verification, debugging, validation, and later modification. It also eliminated a design limitation in the Borland TURBO Pascal package. Program sizes always exceeded the maximum size the Borland product could compile in memory at one time (approximately 20K, depending on code and data sizes). By

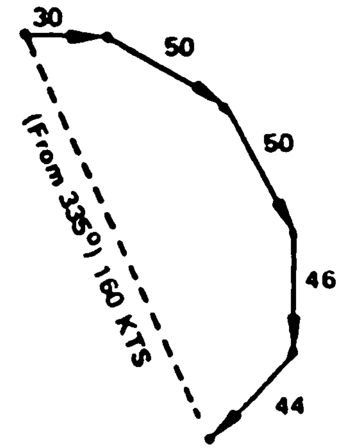
Average Winds Aloft Example

A. WINDS ALOFT DATA

SPECIFIED ALTITUDE	EFFECTIVE RANGE	DIRECTION/ VELOCITY	VECTOR VALUE
	9000		
8000		270/30	090/30
	7000		
6000		300/25	120/50
	5000		
4000		330/25	150/50
	3000		
2000		000/23	180/46
	1000		
SEA LEVEL		045/44	225/44

NOTE: THE VECTOR VALUES FOR 6000, 4000, AND 2000 REPRESENT A RANGE OF 2000 FT. EACH. THEREFORE, THE TOTAL NUMBER OF VECTORS FOR 1000 FT. INTERVALS IS 8.

B. VECTOR SOLUTION



RESULTANT VECTOR:
335/160

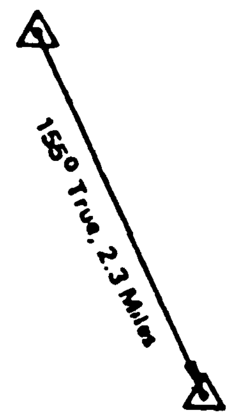
AVERAGE WIND DIRECTION: 335

AVERAGE WIND VELOCITY

$$\frac{160}{8} = 20 \text{ knots}$$

AVERAGE WINDS ALOFT:
335/20

C. PARACHUTE DRIFT



(PARACHUTE OPENED AT 8000 FEET OVER OCEAN)

Figure 5.

chaining modules and compiling to disk instead of memory (options available in the Borland product), no complications arose from the compilation of programs with source code larger than available RAM (64K). This is because modules were individually-compiled to disk instead of simultaneously compiling in memory before recording on disk.

The three programs that make up the search planning package are: Aerospace Drift Determination, Surface Drift Determination, and Search Area Determination. These programs and their modules will now be discussed in more detail (Reader understanding of National SAR Manual search planning methodology is assumed).

Aerospace Drift Determination Program

The Aerospace Drift Determination program has three possible paths of flow (SAR object glides only, parachutes only, or, glides then parachutes to the surface). No matter how it reaches the surface, the search object is displaced from its initial incident position by the vector sum of various wind directions and velocities and different altitudes (see example in Figure 5).

The Aerospace Drift Determination program calls sub-modules in the order shown in Figures 6, 7, and 8.

```
|-AeroSpaceWinds Program
|
|->    |-AeroPosition
|      |
|      |->  VerifyDTG
|      |
|      |->  ChuteData
|      |   (Drift Method = PARACHUTE ONLY; SEE BELOW)
|      |
|      |->  GlideData
|      | (Drift Method = GLIDE & PARACHUTE; SEE BELOW)
|      |
|      |->    |-GlideData
|             |
|             |->    |-AeroGlide
|                    |
|                    |->    |-InputWinds
|                    |      |
|                    |      |->    |-VerifyWinds
|                    |      |      |
|                    |      |      |->  WindChart
|                    |      |
|                    |      |->    |-WindDrift
|                    |             |
|                    |             |->  GlideOrChute
|                    |             |
|                    |             |->  AddVectors
|                    |
|                    |->  AddVectors (Optional)
|
|->  WriteToDisk
```

Figure 6. FIRST PATH: DRIFT METHOD = GLIDE ONLY

```
|-AeroSpaceWinds Program
|
|->    |-AeroPosition
|      |
|      |->  VerifyDTG
|      |
|      |->  GlideData
|      |    (Drift Method = GLIDE ONLY; SEE ABOVE)
|      |
|      |->  GlideData
|      | (Drift Method = GLIDE & PARACHUTE; SEE BELOW)
|      |
|      |->    |-ChuteData
|             |
|             |->    |-InputWinds
|                    |
|                    |->    |-VerifyWinds
|                    |      |
|                    |      |->  WindChart
|                    |
|                    |->    |-WindDrift
|                           |
|                           |->  GlideOrChute
|                           |
|                           |->  AddVectors
|                           |
|                           |->  Ejection
|                           |
|                           |->  AddVectors (Optional)
|
|->  WriteToDisk
```

Figure 7.   SECOND PATH: DRIFT METHOD = PARACHUTE ONLY

```
|-AeroSpaceWinds Program
|
|->    |-AeroPosition
|      |
|      |->   VerifyDTG
|      |
|      |->   ChuteData
|      |     (Drift Method = PARACHUTE ONLY; SEE ABOVE)
|      |
|      |->   GlideData
|      |     (Drift Method = GLIDE ONLY; SEE ABOVE)
|      |
|      |->    |-GlideData
|             |
|             |->    |-AeroGlide
|                    |
|                    |->    |-InputWinds
|                    |      |
|                    |      |->    |-VerifyWinds
|                    |      |      |
|                    |      |      |->   WindChart
|                    |      |
|                    |      |->    |-WindDrift
|                    |             |
|                    |             |->   GlideThenChute
|                    |             |
|                    |             |->   AddVectors
|                    |
|                    |->   AddVectors (Optional)
|                    |
|                    |->   Ejection (Optional)
|                    |
|                    |->   AddVectors (Optional)
|                    |
|                    |->   |-WindDrift
|                          |
|                          |->   GlideOrChute
|                          |
|                          |->   AddVectors
|                          |
|                          |->   AddVectors (Optional)
|
|->   WriteToDisk
```

Figure 8. THIRD PATH: DRIFT METHOD = BOTH GLIDE & PARACHUTE

Aerospace Drift Program Module Descriptions

Aerospace Drift sub-modules accomplish the following tasks:

PROCEDURE	TASK
AddVectors	When given the X and Ycomponents from procedure VectorComponents, this procedure calculates resultant vector bearing (AvgWindFrom) and magnitude (ResultMagnitude).
AeroGlide	Asks user for aerospace object glide information, if applicable.
AeroPosition	Asks user for last known time and position of an aerospace object.
ChuteData	Gathers additional information concerning the aerospace object's parachute descent before commencing calculations.
Ejection	If aerospace object(s) of interest is a (are) pilot(s) or astronaut(s) ejecting from a crippled vehicle, this procedure asks the user for the (judgmental) performance-type of vehicle. Then it sets the ejection distance equal to the standards for that type of vehicle as prescribed in the National SAR Manual (10:8-8).
GlideData	Gathers additional information concerning the aerospace object's glide before commencing calculations.
GlideOrChute	This procedure is used if the aerospace object only glides (or falls), or parachutes (after, or in lieu of, gliding) to the surface. It's purpose is to find the altitudes halfway between each altitude for which a wind direction and velocity have been reported. It begins with the next higher altitude winds reported above the surface level winds, and ends with that mid-altitude that is greater than or equal to the altitude at which the

	descent began. It then uses these mid-altitudes to determine the affective wind component velocities on the descending aerospace object (10:8-10).
GlideThenChute	This procedure is used if the aerospace object glides (or falls) to a lower altitude at which parachute(s) open to slow it's descent to the surface. It's purpose is to find the altitudes halfway between each altitude for which a wind direction and velocity have been reported. It begins with the mid-altitude that is lower than or equal to the parachute(s) opening altitude, and ends with the mid-altitude that is higher than or equal to the altitude at which the descent began. It then uses these mid-altitudes to determine affective wind component velocities on the gliding (or falling) portion of the aerospace object's descent (10:8-10).
InputWinds	Queries user for altitudes at which the aerospace-object begins to fall or glide, or to deploy parachute(s). Then it asks for wind directions and velocities at those altitudes through which the object descends. The user is allowed to verify/change any input data before it is used in determining the resultant drift vector bearing and magnitude.
VerifyDTG	Verifies legitimate date/time/group data input.
VerifyWinds	Allows user to change wind altitudes, directions, and velocities, entered below in procedure InputWinds.
WindChart	Prints keyboard-input wind altitudes, directions, and velocities, on the video screen for user verification.
WindDrift	Determines the resultant drift vector bearing and magnitude.

WriteToDisk	Prints out record of search planning inputs and calculations from this program.

Surface Drift Determination Program

Over time, the search object is displaced from its initial surface/splash-down position by the vector sum of leeway, sea, tidal, and/or wind current directions and velocities at different times and locations (similar to the example presented in Figure 5 above). The Surface Drift Determination program calls sub-modules in the order shown in Figure 9.

```
|-Surface Drift Program
|
|->    |-SurfacePosition
|      |
|      |->  VerifyDTG
|      |
|      |->  VerifyDTG
|
|->    |-WindPeriods
|
|->    |-PeriodTimes
|      |
|      |->  DaysInMonth (Optional #1)
|      |
|      |->  DaysInMonth (Optional #2)
|      |
|      |->  DaysInMonth (Optional #3)
|
|->    |-InputSeaWinds
|
|->    |-WindLatCoeffs
|      |
|      |->    |-VerifyWinds
|             |
|             |->  WindChart
|             |
|             |->  WindChart (Optional)
|
|->    |-WindCurrent
|      |
|      |->  AddVectors
|      |
|      |->  AddVectors (Optional)
|
|->    |-AvgSurfaceWind
|      |
|      |->  AddVectors
|
|->    |-LeewayDrift
|      |
|      |->  DriftDirUncertain (Optional)
|
|->    |-SeaCurrent
|
|->    |-Datum
|
|->    |-WriteToDisk (If program used)
       |
       |-> RecordCurrents
```

Figure 9. SURFACE DRIFT DETERMINATION PROGRAM FLOW CHART

Surface Drift Program Module Descriptions

Surface Drift Determination sub-modules accomplish the following tasks:

PROCEDURE	TASK
AddVectors	When given X and Ycomponents this procedure calculates resultant vector bearing (AvgWindFrom) and magnitude (ResultMagnitude).
AvgSurfaceWind	Determines the average surface (leeway) wind blowing on the search object's exposed area above the ocean's surface.
Datum	Determines the Datum Drift Vector (Min and Max, if applicable) from the vector sum of all previous surface drift vectors.
DaysInMonth	Given the month, determines the number of days in that month, including a check to see if that month is a leap-year-February with 29 days.
DriftDirUncertain	Used by procedure Leeway (if drift rate and time are known with certainty) to calculate minimum and maximum leeway drift direction.
InputSeaWinds	Queries user for ocean surface wind directions and velocities.
LeewayDrift	Calculates the total leeway drift direction and distance (10:8-13 through 8-15).
PeriodTimes	Determines the day and hour for which wind directions and velocities are required in the eight intervals of each period to calculate each period's ocean surface wind current component vector (10:8-16b).
RecordCurrents	Called by procedure WriteToDisk; Prints out record of Wind and Sea Current vectors or Observed Total Water Current vector, as applicable.

SeaCurrent	Determines the sea (or alope) current affecting search object drift, and, queries user for total observed water current vector, if known.
SurfacePosition	Asks user for the last known and the desired Datum times and positions of a search object on the ocean's surface.
VerifyDTG	Verifies legitimate date/time/group data input.
VerifyWinds	Allows user to change ocean surface wind directions and velocities entered in procedure InputSeaWinds.
WindChart	Prints keyboard-input wind coefficient directions and velocities on the video screen for user verification.
WindCurrent	Uses vector addition to calculate the resultant wind current vector direction and magnitude for each period and for the sum of all period vectors.
WindLatCoeffs	Queries user for Wind Latitude Coefficient Vector directions and velocities from <u>National SAR Manual</u> tables (10:8-16c through 8-16d).
WindPeriods	Given the last known surface position time, and the hours elapsed from then until the desired Datum time, this procedure determines the number of 48-hour wind-current-effect periods, and the number of hours (1 to 6) in each period, required to calculate the average ocean surface current caused by the wind.
writelns	Writes out a specified number of blank lines.
WriteToDisk	Prints out record of search planning inputs and calculations from this program.

Search Area Determination Program

The Search Area Determination program calls sub-modules in the order shown in Figure 10.

```
|-Program: SearchArea
|
|-> |-Position
|
|-> |-AeroSurfVectors
|
|-> |-AreaSearch
|
|-> |-FindCoordinates
|    |
|    |-> |-NewCoordinates (Aerospace Drift Vector Pos.)
|    |
|    |-> |-NewCoordinates (Dmax Surface Drift Position)
|    |
|    |-> |-NewCoordinates (Dmin Surface Drift Position)
|    |
|    |-> |-NewCoordinates (Search Area Center Point)
|    |
|    |-> |-FindXYsToCorner
|            |
|            |->  NewCoordinates (Upper Right Corner Pos.)
|            |
|            |->  NewCoordinates (Lower Left Corner Pos.)
|
|-> |-WriteToDisk
```

Figure 10. Search Area Determination Program Flow Chart

Search Area Program Module Descriptions

Search Area Determination sub-modules accomplish the following tasks:

PROCEDURE	TASK
AreaSearch	Given the search object's aerospace and/or surface drift vector(s), this procedure calculates the search area.
AeroSurfVectors	Queries user for previously-calculated Aerospace and (Max & Min) Surface Drift vectors.
FindCoordinates	Calculates X & Ycomponents of aerospace and surface drift vectors and sends them to procedure NewCoordinates for it to determine the search area center point latitude/longitude. Next, this procedure creates two vectors (1 & 2), 90-degrees apart, and of length equal to the search area's radius. It sends these two vectors to procedure FindXYsToCorner for it to calculate the vectors' X & Ycomponents enroute to procedure NewCoordinates, which finds the corner point latitudes/longitudes.
FindXYsToCorner	Finds the X & Ycomponents of the search-area-center-point-displacement-vectors created by the calling procedure, FindCoordinates. Then, it passes these components to procedure NewCoordinates for it to determine search area corner point latitudes/longitudes.
NewCoordinates	When given X & Ycomponents and a reference latitude/longitude by the calling procedures FindCoordinates and FindXYsToCorner, this procedure calculates the updated position's latitude/longitude.
Position	Queries user for last known position latitude and longitude.

writelns	Writes out a specified number of blank lines.
WriteToDisk	Prints out record of error adjustments to calculations of search radius and area.

Chapter 3

Program Structure

Overview

Chapter 2 declared the attached programs are designed using current software engineering guidelines. This chapter discusses these guidelines and how they were used to make this search planning software accurate, maintainable, reliable, and "user-friendly" (for users knowledgeable in search planning methodology).

Software Engineering Guidelines

According to Ledgard in Pascal With Style: Programming Proverbs, the most important guideline in software development is the *use of top-down design* using modules:

> Top-down programming has two distinct advantages. First, a programmer is initially freed from the confines of a particular language and can deal with more natural data structures or actions. Second, it leads to a modular approach that allows the programmer to write statements relevant to the current structures or actions. (3:13)

The modules are then thoroughly-tested before their combination into the larger program. The final step involves program verification (testing) to pinpoint module interface errors.

Testing & Error Checking

Unfortunately, program testing only reveals the presence, not the absence, of errors:

> Too often quantity of test data is accepted in place of quality. The mere fact that a program works on 50 test runs means nothing if those 50 sets were not chosen carefully. . . (The) objective must be to select values that cause the execution of all flow paths through the program. (6:436)

In addition, the testing sequence should include boundary, null, and illegal case trials.

Boundary case errors are uncovered by the input of data which falls at the boundaries or extreme legal range allowed by the program. For example, a calendar-date program should be tested for accuracy in determining the day after the last day of different months, last day of the year, and last day of a leap year February (as is done by Procedure DaysInMonth in the attached Surface Drift program).

In contrast, testing for null case error involves such tricks as the input of blanks or carriage returns where data is requested by the program, or directing the program to reference an empty data file. Such actions should not cause an abrupt abort of the program run, although wrong answers will be generated. To avoid null errors, the attached programs are initialized with valid values, and, they often lock the user into a "repeat-until" loop until valid new data is input. Also, there are no calls to external data files, empty or otherwise.

Finally, programs must be checked for their "robustness" in handling illegal data entry. A robust

program produces meaningful results (although not always correct answers) from any input data set, even if that set is illegal or improper. Since the three programs created for this project request numerous data inputs from the user, input errors are a certainty. The type of errors expected are:

(1) Accidently typing the wrong character.

(2) The input of a real number where an integer is requested.

(3) The input of wind vector data out of sequence.

(4) Transposing the input of a date-time-group (e.g., 1200-25Z 1/86 vs. 251200Z JAN 1986).

Improper values accepted by these programs leads to the "Garbage-In, Garbage-Out" (GIGO) syndrome. To prevent such an occurrence, "defensive programs" have been designed with extensive error-checking for legal and plausible data values. When data is requested from the user, format examples are provided. Then, user input is filtered through "repeat-until" loops to ensure adherence to required format. For example, the programs will not accept compass headings outside of the 0 to 360 degree range, winds greater than 99 knots, or selection of option #4 when only three choices are offered. In some cases, the programs will print an appropriate error message, explaining how to correct and resubmit the data.

Unfortunately, the design of the Pascal programming language allows erroneous input of real numbers where integers are called for. This action leads to an extremely abrupt program abort. All data must then be re-input from the beginning. However, since the search planning package was divided into three separate programs, data re-input and execution time of any one is usually less than a minute for experienced users with the required data readily-available.

Calculation Accuracy

Another problem over which there is no control is the machine representation and manipulation of real numbers. This leads to incorrect fractional values and rounding or truncation errors. For example, a computer having eight binary place accuracy records the value "1/5" as 0.00110011, or 0.1992 instead of 0.20. In this case, five times a fifth equals 0.996! Therefore, checking for equality of real values or successively multiplying or dividing them introduces errors during program execution. Steps were taken in designing the attached search planning programs to minimize opportunities for the occurrence of these errors.

Summary Of Other Design Features

Other design features of attached programs include:

(1) Cleverness of expression was always sacrificed for expression clarity and simplicity.

(2) Mnemonic variable names correspond with those

used by the USCG National SAR School, even at the expense of having some names 18 characters in length.

(3) Variable prefixes and suffixes are standardized (e.g., different types of wind's directions, velocities, or distances covered, always use the terms "Brg" (Bearing to), "Dir" (Direction from), "Spd" (Speed), or "Dist" (Distance) in their mnemonic name).

(4) Comments to assist program readability and modification are used extensively:

- Comments just prior to procedure declaration describe their purpose.
- All variables names and functions are comment-defined in the appropriate "Var" and "Const" sections.
- All "Begin" and "End" pairs are matched using comments.
- Any statement whose intent is not obvious is described.

(5) To enhance program clarity:

- Each line is numbered (non-standard for Pascal).
- Only one statement per line is allowed (except for variable value initialization).
- Multiline statements are indented so that structurally-related clauses are aligned.

(6) Following each of the three source codes in Appendix D are tables of the mnemonic variables and Pascal functional operators used by each program. These tables

also include a complete, cross-referenced listing of line numbers on which each variable and operator appears. These "Variable & Operator Cross-Referenced Listings" are provided to further enhance software understanding and aid modification efforts.

(7) All input wind altitudes, directions, and velocities are reprinted back to the user for verification and update, if necessary, before being used in program calculations. The only exception are those wind current wind times, directions, and velocities requested by the largest of the three attached programs -- Surface Drift Determination. Unfortunately, there was only enough microcomputer memory space remaining in the Surface Drift program to verify this data, or to verify the requested Wind Latitude Coefficient Vector directions and magnitudes. Since the Latitude Coefficient data is more complex, it appears the likeliest candidate for typographical input error. For this reason, it was selected to fill the remaining verification space. Therefore, the user must exercise caution to carefully input wind current wind data the first time. Any errors will require the user to terminate that program run and try again.

(8) The attached search planning programs request some input from the user that is not fully tested for error. An example is the query for the search object's last known position date-time-group (DTG) in the first program --

Aerospace Drift Determination. Although the DTG format is checked to confirm the date falls between 1 and 31 and the hour between 0000 and 2300, the date is not correlated with the month. For example, "311400Z Feb 1984" (or "2:00 P.M. 31 Feb 1984" -- 31 days in February!) is entirely acceptable by the program. Why? The DTG is not used in any calculations in this particular program, but only for final output file reporting purposes. Therefore, scarce microcomputer memory space was conserved by relying on the user to avoid logic or typographical input errors, while maintaining the his/her ability to recognize such simple errors, should they occur in the final output disk file.

(9) "Goto" statements were not used to avoid unconditional transfers of control thereby enhancing program readability.

(10) The use of global vs. local variables was abused in designing the attached programs. Why? Calling variables "by location" uses much less of the scarce microcomputer memory than "call by value" routines. Also, after all user input and computations are completed, most program variables must be written to an external record/file to serve as a "calculations-audit trail." Making these variables global allowed for a simple, "WriteToDisk" procedure to create the required record.

(11) All three attached programs take advantage of a non-standard Pascal feature of Turbo Pascal -- a built-in

command/procedure called "CLRSCR" for clearing the video screen whenever desired. For these programs to run properly under standard Pascal rules, these "CLRSCR" commands should be globally replaced with the command "writelns(24)". This command calls on the predefined procedure Writelns included in all three programs. The unit "24" informs this procedure to print 24 blank lines on the output device (monitor or printer), essentially clearing the screen just as "CLRSCR" (although not as briskly).

<u>Validation/Verification</u>

The goal of verification and validation was to test the programs to ensure they perform as expected, providing valid and correct output from legitimate input data. Verification of the attached programs was accomplished in two steps.

Individual modules were first tested and re-tested during the design phase to guarantee they accurately performed only those functions they were created to do. Input and output test data came from two scenarios provided by the Coast Guard. Once the modules passed these preliminary inspections, they were combined to form the overall program. This program was then subjected to additional testing to verify it still worked as desired -- grinding out identical answers to those supplied by USCG.

Confirming program "correctness" under a variety of test conditions, or "validation," was generously

accomplished by the Coast Guard. The attached programs were downloaded to their Convergent Technologies C-3 Data System mainframe computer system in New York in September of 1984. Then, National SAR School faculty members ran numerous school and real-world problems to test program limits and accuracy. The result -- in each case the search planning software performed flawlessly, generating answers identical to those expected.

Chapter 4

Epilogue

Conclusions

The initial intent of this thesis effort was to create fast, reliable, and accurate, microcomputer-based, search planning software to be used to maintain the skills of those SAR planners/National SAR School graduates patient enough to retype (and compile) the source code from free copies provided by the SAR School. From this meager beginning, the three programs generated by this effort developed a life of their own.

The great breakthrough came with the creation of an electronic pathway between the C-3 mainframe computer (used at almost every Coast Guard base) and the Osborne microcomputer during the author's visit to USCG Governor's Island, New York. This allowed transfer and subsequent compilation of the search planning software, with the inherent advantages of:

(1) National SAR School faculty members had around-the-clock access on their own office terminals to the search planning software created by this thesis effort. This access enabled them to verify and validate this software, as well as make valuable modification recommendations before the final version(s) of the three programs were released. (2) Search planners/National SAR School graduates would not be forced to retype the +65-pages of source code (which

would certainly discourage widespread dissemination of the software). Instead, they could now download the source code directly to disk (in seconds) or via modem telecommunications to their microcomputers (in minutes).

(3) The first of the three attached programs, Aerospace Drift Determination, suddenly achieved great significance upon the discovery that the Coast Guard's commercially-procured, surface search planning software (CASP) lacks the crucial capability to determine aerospace drift (glide, fall, and/or parachute winds) vectors. USCG National SAR School faculty members spoke of using this program for more than just refresher training purposes.

The only unavoidable complaint directed against the attached software was actually a *criticism* of the programming language used. SAR School faculty members disliked the abrupt way in which programs aborted when a fatal Pascal error was input (e.g., input of a character where an integer was requested -- this particular problem was corrected by module redesign and a change of input variable type).

Recommendations For Further Research

None. This is not to assert the design of the attached software could not be improved. However, such embellishment could easily exceed scarce microcomputer memory limits, thus violating the prime objective of this entire effort.

The only feature which could be added to the attached package is search area "Effort Allocation" -- dividing the search area into sub-areas suited to the capabilities and duration of each available search craft and having the highest probability of containing the search object. However, optimal effort allocation involves the capability of weighted multi-criteria decision making and geographic map generation, both of which are better suited for mainframe computer implementation due to the large amounts of memory and processing required. In fact, the Coast Guard's mainframe CASP program does both in an effort allocation program far beyond the reach of current microcomputer capability (8).

Bibliography

1. "Cosmos Reentry Spurs Nuclear Waste Debate." Aviation Week & Space Technology Magazine, January 30, 1978, pp. 33.

2. Fraley, Major Theodore R. E., USAF, and Captain Dale A. Kem, USA. Fortran Based Linear Programming For Microcomputers, MS Thesis GOR/OS/82D-4. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1982 (AD-A124 804).

3. Ledgard, Henry F., John F. Hueras and Paul A. Nagin. Pascal With Style: Programming Proverbs. Rochelle Park, NJ: Hayden Book Co., Inc., 1979

4. Masterson, Fred A. "Languages For Students." Byte Magazine, June, 1984, pp. 233-238.

5. "NASA Abandons Skylab Control Plans." Aviation Week & Space Technology Magazine, January 1, 1979, pp. 19-20.

6. Schneider, G. Michael, and Steven C. Bruell. Advanced Programming And Problem Solving With Pascal. New York: John C. Wiley & Sons, Inc., 1981.

7. "Soviet Cosmos 954 With Nuclear Reactor Aboard Reenters Earth's Atmosphere January 24 - Disintegrates Over Northwestern Canada." New York Times, January 25, 1978, 1:6.

8. Telephone conversations, course materials, and personal interviews with faculty members of the United States Coast Guard National Search and Rescue School, USCG Governor's Island, New York, during the period Feb 84 through Oct 84.

9. "US-Canadian Air-Search Teams Discover Powerful Radiation In Area Where Satellite Reentered Atmosphere." New York Times, January 27, 1978, 1:5.

10. U.S., Department of Defense, Department of the Air Force. Air Force Manual 64-2*, National Search and Rescue Manual (*Also: USA FM 20-150, USN NWP-37(B), and USCG COMDINST M16130.2). Washington, D.C.: Government Printing Office, July 1, 1973, Amendments #1 through #7.

11. "Use of Pascal Language in Training of Computer Programmers." Sovetskaya Litva, No. 167 (12480), July 20, 1984.

Appendix A

DEFINITION OF TERMS

Glossary

AEROSPACE ASSET - Any object of value which operates in or passes through the Earth's atmosphere, e.g. aircraft, spacecraft, debris of same, pilots, astronauts, etc.

AEROSPACE DRIFT - The ground distance an aerospace asset could possibly cover during its descent.

AEROSPACE POSITION - The initial position of an aerospace asset at the time of reentry, engine failure, or aircrew ejection/bailout

BAILOUT TRAJECTORY - Momentum imparted in the direction of travel by an aircraft movement upon an airman who has ejected or bailed out.

CONFIDENCE FACTOR - Values ranging from 0.125 to 0.375 used to obtain drift error. Confidence factor is the search planners estimate of how reliable his/her information is in terms of initial distress position, sea current, wind current, and leeway. If one or more of these factors are poorly defined, confidence is low and a value of 0.30 is used. If all four factors are known with high certainty, a value of 0.125 is used.

CRASH - A landing in which the vertical velocity is so great and the time spent in reducing it to zero is so brief that the acceleration and hence the forces acting become so great as to result in structural failure.

DATUM - The probable location of the search object corrected for drift at any particular moment during the mission.

DATUM MINIMAX - The datum point established midway between the resultant minimum drift position and the resultant maximum drift position.

DEAD RECKONING - Determination of one's position by monitoring course and adding the additional displacement since passing last position. This displacement is calculated by multiplying one's speed by the time elapsed since passing last position.

DRIFT - The vectorial movement (direction and distance) of the search object caused by momentum, drag, wind, water, or other external forces.

HEADING - The horizontal direction in which an aerospace or surface asset is pointing/moving.

INDIVIDUAL DRIFT ERROR - Individual drift multiplied by Confidence Factor.

INITIAL POSITION ERROR - The assumed error of the initially reported position of a search and rescue incident.

KNOT - A measurement unit of speed equivalent to one nautical mile per hour.

LEEWAY - The movement of a search object caused by being pushed through the water by local winds acting against the exposed surface of the object. This motion is distinct from the effect of the wind on the surface current (see **WIND CURRENT** on next page).

LAST KNOWN POSITION - The position last witnessed, reported, or computed as a dead-reckoning position from a previously-reported and reliable position of the search object.

PARACHUTE DRIFT - The combined drift of parachute glide ratio and its displacement due to winds aloft while the parachute is descending.

SAFETY FACTOR - A factor used to increase Total Probable Error to a length that insures greater than a 50% probability that the object is in the calculated search area.

SEA CURRENT - 1. Current present in the open sea caused by factors other than local winds (combined with wind current to form total water current in oceanic areas); 2. The permanent, large scale flow of ocean waters, also called "slope current." Not usually computed for an incident of less than four hours of drift, unless the target is in a relatively high speed current area such as the Gulf Stream.

SEARCH AND RESCUE/RECOVERY - The employment of available personnel and facilities in rendering aid to persons and property in distress.

SEARCH AREA - That area calculated to contain the search object and subject to intensive scrutiny. Because there are few search patterns which are easily adaptable to a circular search area, a square is circumscribed around the first search circle. Thus a square search area with sides equal to twice the search radius and centered at datum is established.

SEARCH CRAFT ERROR - Error introduced by the less than 100% navigational accuracy of equipment installed aboard specific types of search craft.

SEARCH RADIUS - The radius of a circle centered on a datum point. It has a length equal to the Total Probable Error plus an additional safety length to insure greater than 50% probability that the object is in the search area.

SET - The compass direction towards which a current flows, or the direction an object moves under the influence of wind and/or current.

SURFACE DRIFT - The vector sum of the average sea current, wind driven current, and leeway.

SURFACE POSITION - The surface position (in latitude and longitude) of the search object on the Earth's surface.

TIDAL CURRENT - Water currents influenced predominantly by reversing, coastal rotary, or river currents, or their combinations.

TOTAL DRIFT ERROR - The sum of all individual drift errors during a certain time interval.

TOTAL PROBABLE ERROR - That error in datum within which it is assumed there is a 50% chance that the search object is located. It is the square root of the sum of the squares of the Total Drift Error, the Initial Position Error, and the Search Craft Error.

VECTOR - A quantity having both direction and magnitude.

WIND BEARING - The compass direction toward which a wind is blowing.

WIND CURRENT - The current generated by the wind acting upon the surface of the water for a period of time, affected by the wind's velocity and duration.

WIND DIRECTION - The compass direction from which a wind is blowing.

Acronyms

AFIT - Air Force Institute of Technology

AGL - Above Ground Level (altitude)

AFM - Air Force Manual

CASP - Computer-Assisted Search Planning

COMDINST - USCG Commandant's Instructions (regulations)

Dmax - maximum Drift distance

Dmin - minimum Drift distance

DTG - Date-Time-Group (e.g. "291000Z Jul" is equivalent to July 29th at 1000-hours zulu time.)

ETA - Estimated Time (of) Arrival

FM - (USA) Field Manual

GIGO - "Garbage-In, Garbage-Out" (computer proverb)

KM - Kilometer

LKP - Last Known Position

MINIMAX - Point halfway between maximum and minimum

MSL - Mean Sea Level (altitude)

NAUT - Nautical mile

RAM - (Computer) Random Access Memory (space)

SAR - Search and Rescue/Recovery

SSDD - Single-Sided, Double-Density (magnetic disk)

TAC - (USAF's) Tactical Air Command

USA - United States Army

USAF - United States Air Force

USCG - United States Coast Guard

USN - United States Navy

Xcomponent - The horizontal component of a vector used in vector addition

Ycomponent - The vertical component of a vector used in vector addition

Appendix B

SOFTWARE USER'S GUIDE

Overview

The purpose of this guide is to provide all the information a user needs to employ the attached aerospace asset search planning programs properly. The user must have a microcomputer equipped with a minimum of 64K RAM and one disk drive. The three programs generate input/output data record files during run-time which easily fit any standard 80-column-width printer at ten characters-per-inch (no "line wrap-around"). These search planning programs are recommended for training use only, especially if human life is at risk. Also, due to the possibility of undetected errors, user's should be totally familiar with search planning methodology before attempting to use this software (completion of the USCG National SAR School search planning course is highly-encouraged).

Software Purpose

The attached programs provide the user with the necessary automated tools to complete search area planning for missing persons or aerospace objects in the Earth's most complex drift environment -- the ocean. The first program calculates drift forces on a falling, gliding, or parachuting aerospace object to determine the displacement vector from its last known coordinates. The second program

calculates wind current, sea current, and leeway drift forces on a surface object to determine the maximum and minimum displacement vector from its last known position. Finally, the third program combines previously input data and computed vectors to determine the center and four corner points of the search area with the highest probability of containing the missing person(s) or object.

Data Required

To calculate these drift vectors and the search area, the programs request specific data inputs from the user. Each program requests different data with some duplication for continuity. A detailed description of the input data required for each of the three attached programs follows.

Aerospace Drift Program Inputs

In the Aerospace Drift Program the user will be asked for:

(1) Last known aerospace position and time (month, year, date-time-group, latitude, and longitude);

(2) Surface altitude or terrain height;

(3) Altitude (Incident altitude) at which the object began it's glide, fall, and/or parachute deployment, as applicable;

(4) Altitude lost during glide, fall, and/or parachute deployment, as applicable;

(5) Object's glide ratio, descent heading and rate,

as applicable;

(6) Wind directions and velocities at every reported level up to incident altitude (plus one or two more above it) with the opportunity to verify or change the input before committing it to internal calculations;

(7) Parachute drift distance and chute descent rate from National SAR Manual tables, if applicable (10:8-11 through 8-12); and,

(8) Whether the aerospace object was a medium- or high-performance jet aircraft (as defined by the National SAR Manual) to determine pilot/crew ejection distance, if applicable (10:8-8).

Once the run is complete, the Aerospace Drift program creates an external text file ("calculation audit trail") named "AERODATA" detailing user inputs and program outputs.

Surface Drift Program Inputs

In the Surface Drift Program the user will be asked for:

(1) Last known surface position and time (month, year, date-time-group, latitude, and longitude);

(2) Desired (Datum) surface search time (month, year, date-time-group);

(3) To confirm the search object is more than 20 miles (32 km) off shore in water greater than 100 feet (32 m) in depth as prescribed for this search planning method by the National SAR Manual (10:8-16a);

(4) Approximate hours elapsed from last known position time to search initialization;

(5) Sea wind directions and velocities (reported at 0000, 0600, 1200, and 1800 Zulu time) for at least a 48-hour period from last known position time to search initialization time (User must exercise caution to input this particular data correctly the first time. It is not echoed back for user verification and update due to scarce microcomputer memory limits. If an input error is made, followed by a carriage return, the user must manually abort and rerun the program to obtain reliable calculations.);

(6) Wind latitude coefficient directions and magnitudes for the appropriate latitude from the National SAR Manual (10:8-16c through 8-16d) with the opportunity to verify or change the input before committing it to internal calculations;

(7) Choose from drift rate, drift time, or directional drift uncertainties. Then input minimum and maximum leeway drift speeds, minimum and maximum hours of drift, and/or the maximum expected drift divergence angle (from the National SAR Manual), as applicable (10:8-15);

(8) Sea current drift direction, velocity, and published source of information (10:8-16i through 8-16j);

(9) Total observed water current, if applicable; and,

(10) Tidal current direction and velocity, if applicable (10:8-22).

Once the run is complete, the Surface Drift program creates an external text file ("calculation audit trail") named "SEADATA" detailing user inputs and program outputs.

<u>Search Area Determination Program Inputs</u>

In the Search Area (Determination) Program the user will be asked for:

(1) Last known surface coordinates (latitude and longitude);

(2) Previously-calculated total aerospace drift direction and distance, if applicable;

(3) Aerospace drift error confidence factor from the <u>National SAR Manual</u> (10:8-27);

(4) Sum of previously-calculated drift errors, if applicable;

(5) Previously-calculated minimum and maximum total surface drift direction and distance;

(6) Plotted distance between these surface drift positions;

(7) Surface drift error confidence factor from the <u>National SAR Manual</u> (10:8-27);

(8) Search object and search craft navigational fix and dead-reckoning errors from the <u>National SAR Manual</u> (10:8-29); and,

(9) Whether this is the first through the fifth search effort (to determine total probable error factor) (10:8-32a through 8-32b).

Once the run is complete, the Search Area program creates an external text file ("calculation audit trail") named "AREADATA" detailing user inputs and program outputs.

<u>Software Maintenance Responsibility</u>

The author makes no express or implied warranty of any kind with regard to the attached programs. This includes, but is not limited to the implied warranty of fitness for any particular purpose. The author shall not be liable for incidental or consequential damages in connection with or arising out of the furnishing, use, or performance of these programs.

Published changes to the <u>National Search and Rescue Manual</u> or errors detected by users should be brought to the attention of the author:

Captain D. R. Douglas, USAF
2065 East 3300 South
Salt Lake City, UT 84109 ,

who will make every effort to update or correct the attached software and re-release it to public domain under a different version number. The attached "Microcomputer Application of Aerospace Asset Search Planning" programs are released as versions 1.0 or 1.1.

Appendix C

SAMPLE PROBLEM / SOFTWARE DEMONSTRATION

Overview

This section introduces a sample search planning problem from the USCG National Search and Rescue School. The data presented is then input into the attached programs and solutions generated to demonstrate software execution. Appearing in brackets ( ) beside some of these program solutions are answers provided by the Coast Guard, as well as a percentage error difference between the two solutions due to the previously-described internal rounding errors, etc. . For example:

Program solution	USCG solution
X = 0.950	(X = 0.974; 2.464% error)
Y = 0.892	(Y = 0.887; 0.0056% error)

Sample Problem Introduction

Situation: At 1000Z, 29 April 1984 a distress call was received from an ASAT-equipped (anti-satellite weapon), Air Force F-15 jet fighter. The pilot advised he was ejecting at 10,000 feet AGL (above ground level) in position 39-15.0 North 64-30.0 West. At time of bailout, the pilot's high-performance aircraft was on an satellite interception heading of 360-degrees Magnetic (348-degrees True, accounting for magnetic variation of 12-degrees West at that location). According to the parent command (TAC), F-15

ejection seats are equipped with self-deploying, single-man, orange life rafts with built-in drogue (sea anchors).

The first available search craft, an Air Force HC-130H from the Aerospace Rescue and Recovery Service, will arrive on scene at approximately 291900Z Apr 1984 to begin the search. A Russian trawler has also signalled intentions to divert and provide assistance with an ETA (estimated time of arrival) in the incident area of 300600Z.

The following meteorological data has been obtained for the vicinity of 39-15N 64-30W:

291000Z UPPER WINDS FORECAST FROM WSFO, JFK, NEW YORK

Surface 070/15 (070-degrees at 15 knots)

3000	120/30	18000	210/25
6000	130/25	24000	210/20
9000	170/48	30000	210/30
12000	165/52	34000	215/65

SURFACE WIND HISTORY/FORECAST FROM USAF AIR WEATHER SERVICE

270000Z	100/15	290000Z	040/25
0600Z	070/18	0600Z	065/20
1200Z	065/10	1200Z	074/20
1800Z	080/20	1800Z	077/18
280000Z	040/30	300000Z	090/10
0600Z	035/40	0600Z	090/20
1200Z	040/40	1200Z	065/15
1800Z	045/20	1800Z	050/20

USN Publication 1400-NA 6 for the approximate position shows sea current set = 071-degrees True at 1.1 knots.

Directional drift uncertainty is assumed with a leeway drift divergence of plus-or-minus 35-degrees (from the National SAR Manual).

Use the attached programs to determine:

(1) Average winds aloft
(2) Aerospace drift vector
(3) Average Surface winds
(4) Wind current vector
(5) Leeway vector
(6) Maximum & minimum datum vectors
(7) Search area size
(8) Search area center point coordinates
(9) Search area corner point coordinates

<u>Sample Problem Program Runs</u>

```
<< RUN FIRST PROGRAM: A E R O S P A C E   D R I F T >>
/\/\/\/\/\/\/   N E W   S C R E E N   \/\/\/\/\/\/\

(*************************************************************************)
(*          SEARCH PLANNING SOFTWARE (PROGRAM #1 OF 3)                   *)
(*  TITLE:          AERODRIF.COM (Aerospace Drift Algorithm)             *)
(*  VERSION:        1.1  for CP/M Operating System                       *)
(*  DATE WRITTEN:   September  1984                                      *)
(*  LICENSE:        COPYRIGHT 1984     D. RICK DOUGLAS                   *)
(*************************************************************************)
The author makes no express or implied warranty of any kind with regard to
this program material, including, but not limited to, the implied warranty of
fitness for a particular purpose. The author shall not be liable for
incidental or consequential damages in connection with or arising out of
furnishing, use, or performance of this program. The reader MUST HAVE a solid
understanding of search and rescue methodology before using this software in
making decisions where human life is at risk. In fact, since no amount of
testing can uncover 100% of program errors, this program is recommended for
training use only. Prior attendance at the United States Coast Guard's
National SAR School is highly-encouraged.

(***************************** WARNING! **********************************)
(*     THIS SOFTWARE MAY BE FREELY-DISTRIBUTED PROVIDED NO FEE           *)
(*        IS CHARGED AND THIS COPYRIGHT NOTICE IS RETAINED               *)
(*************************************************************************)

PLEASE HIT RETURN (Once or twice, as necessary) TO CONTINUE
```

```
  /\/\/\/\/\/\/     N  E  W    S  C  R  E  E  N     \/\/\/\/\/\/\

This program calculates the initial surface position coordinates
(latitude/longitude) of a falling, gliding, or parachuting
aerospace object or person.  If these coordinates are all ready
known, or the search object did not fall at least 500 feet, then
enter N to the next question and proceed with other search
planning as outlined in the National Search and Rescue Manual.

  Do you wish to continue with this program?  (y/n) => Y


  /\/\/\/\/\/\/     N  E  W    S  C  R  E  E  N     \/\/\/\/\/\/\

Please enter the number for the month the search object was
at the last known position:

 1 = Jan         4 = Apr         7 = Jul         10 = Oct
 2 = Feb         5 = May         8 = Aug         11 = Nov
 3 = Mar         6 = Jun         9 = Sep         12 = Dec

MONTH=> 4

Please enter the year the search object was at the last known
position  (e.g., 1985, 1986, etc.)
YEAR=> 1984


Please enter the day-hour-minute the search object was at the
last known position.  Enter it in Z DTG (Zulu Date-Time-Group)
format.  For example: 9:37 PM, August 4, would appear as
042137 Greenwich-Mean Time (Z=0).
TIME (Z DTG)=> 291000

Was search object's last known latitude north or south?
(Enter N or S)      Answer=> N

Please enter the search object's last known latitude
(For example: 25-Degrees 45-Minutes 13-Seconds = 25.4513)
LATITUDE => 39.15

Was search object's last known longitude east or west?
(Enter E or W)      Answer=> W

Please enter the search object's last known longitude
(For example: 160-Degrees 45-Minutes 13-Seconds = 160.4513)
LONGITUDE => 64.3
```

Please enter the surface level altitude or terrain height (in feet) above or below mean sea level. Enter a zero if altitude equals sea level, or, enter a negative altitude if below sea level (e.g. Death Valley, California.).
SURFACE LEVEL/TERRAIN HEIGHT => 0

After receiving report of the search object's last known coordinates (lat/long), did the object fall or glide to a lower altitude before parachute(s) opened or ground contact was made? (y/n) => N

After receiving report of the search object's last known coordinates (lat/long), did the object successfully use parachutes to descend? (y/n) => Y

Approximate (to the nearest 500 feet) at what altitude above mean sea level (MSL) did the parachute(s) open?
(Enter a zero if descent began below 500 feet AGL)
ALTITUDE=> 10000

/\/\/\/\/\/\/ N E W S C R E E N \/\/\/\/\/\/\

PLEASE ENTER REPORTED OR FORECAST WIND DIRECTION AND VELOCITY FOR EACH KNOWN ALTITUDE WITHIN THE FOLLOWING CONSTRAINTS:

1. You must input the altitude, wind direction (0 to 360), and velocity (0 to 99), from surface level up to the altitude at which the power-off descent began or parachute(s) opened, whichever is higher. You will then be asked to enter one higher altitude for which wind data is known. Finally, you will be able to verify your input data before any computations are made.

2. Altitudes must be entered in rounded thousands of feet, e.g., 3000, 7000, 12000, not 3500, 5200, etc.

3. An example follows of input altitude/winds data format:

```
ALTITUDE       =  0 (Surface level); Enter lowest altitude first!
WIND DIRECTION = 330
WIND VELOCITY  = 25
```

HIT RETURN (Once or twice, as necessary) TO CONTINUE

```
   /\/\/\/\/\/\/    N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\


Begin altitude/wind data input starting with surface level winds:
ALTITUDE       = 0
WIND DIRECTION = 70
WIND VELOCITY  = 15

ALTITUDE       = 3000
WIND DIRECTION = 120
WIND VELOCITY  = 30

ALTITUDE       = 6000
WIND DIRECTION = 130
WIND VELOCITY  = 25

ALTITUDE       = 9000
WIND DIRECTION = 170
WIND VELOCITY  = 48

ALTITUDE       = 12000
WIND DIRECTION = 165
WIND VELOCITY  = 52

Enter data for just one more, higher altitude.

ALTITUDE       = 18000
WIND DIRECTION = 210
WIND VELOCITY  = 25

   /\/\/\/\/\/\/    N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\


 1.  Winds at      0 feet:   70 degrees at  15 knots
 2.  Winds at   3000 feet:  120 degrees at  30 knots
 3.  Winds at   6000 feet:  130 degrees at  25 knots
 4.  Winds at   9000 feet:  170 degrees at  48 knots
 5.  Winds at  12000 feet:  165 degrees at  52 knots
 6.  Winds at  18000 feet:  210 degrees at  25 knots

Are these altitudes, wind directions, and wind
velocities all correct? (y/n) T  (INPUT ERROR (INTENTIONAL))
                                 (CAUSES QUESTION TO REPEAT)

Are these altitudes, wind directions, and wind
velocities all correct? (y/n) Y
```

/\/\/\/\/\/\/\/ N E W S C R E E N \/\/\/\/\/\/\/\

Wind Data (Glide or Parachute Opening Altitude to surface level AGL)
With Velocity Components Calculated Prior To Vector Addition:

```
  Winds at  18000 feet:  210 degrees at   0 knots
  Winds at  12000 feet:  165 degrees at   0 knots
  Winds at   9000 feet:  170 degrees at  120 knots
  Winds at   6000 feet:  130 degrees at  75 knots
  Winds at   3000 feet:  120 degrees at  105 knots
  Winds at      0 feet:   70 degrees at  15 knots
```

```
Xcomponent =  183.319   ( 183.319; 0% ERROR)
Ycomponent =  -213.756  (-213.756; 0% ERROR)
```

The average winds aloft affecting the parachute drift of the aerospace object are bearing to 319 degrees at 28.2 knots.

Refer to the "Parachute Drift Table" in the National Search and Rescue Manual (approximately page 8-12) and interpolate parachute drift distance (in nautical miles) from the published chart. For example: For a parachute opening altitude of 3000 feet (915 meters) and winds aloft averaging 15 knots, the parachute drift distance interpolates to 0.675 nautical miles).
PARACHUTE DRIFT DISTANCE => 3.94

Refer to the "Parachute Descent Data Table" in the National Search and Rescue Manual (approximately page 8-11) and determine the parachute descent rate (in feet-per-second) from the published chart. For example: A 28-foot (C-9) escape parachute at 7000 feet lowers a person at a rate of 21.4 ft/sec, while the three, 83-ft.-diameter parachutes on the Apollo space capsule allowed for a 32.5 ft/sec descent at this same altitude.
PARACHUTE DESCENT RATE => 21.4

Please enter descent or bailout heading, if known
(Enter 361 if not known). HEADING (0-361) => 348

If aerospace object is a pilot ejecting from a crippled aerospace vehicle, was that vehicle:

1. A turboprop or medium performance jet aircraft
2. A high-performance jet aircraft
3. Neither of the above

ENTER A 1, 2, or 3 => 1

```
Xcomponent =  -2.669   (-2.671; 0.0075% ERROR)
Ycomponent =  3.480    ( 3.481; 0.0029% ERROR)
```

```
/\/\/\/\/\/\/    N E W   S C R E E N    \/\/\/\/\/\/\
```

A record of significant input and output data used during this program run is stored in an external file named "AERODATA". If you desire to keep this record permanently, please rename file AERODATA before running this program again!

```
<<        F I R S T   P R O G R A M   O U T P U T        >>
/\/\/\/\/\/\/    N E W   S C R E E N    \/\/\/\/\/\/\
```

```
Missing aerospace object/pilot(s) last known position: 39-15 North  64-30 West
Time: 2910002  4/1984
Bailout/Parachute opening altitude     =  10000 feet MSL
Surface level or terrain height        =      0 feet MSL
Wind bearing affecting para-descent    =    319 degrees True
Wind velocity affecting para-descent   =   28.2 knots
Parachute table descent rate           =   21.4 feet per second
Parachute drift table distance         =    3.9 nautical miles
Ejection displacement                  =    0.5 nautical miles
Total aerospace drift bearing          =    323 degrees True
Total aerospace drift distance         =    4.4 nautical miles
Total time of parachute descent        =    7.8 minutes

  Wind Data Used To Calculate Above Results:
 1.  Winds at      0 feet:   70 degrees at  15 knots
 2.  Winds at   3000 feet:  120 degrees at  30 knots
 3.  Winds at   6000 feet:  130 degrees at  25 knots
 4.  Winds at   9000 feet:  170 degrees at  48 knots
 5.  Winds at  12000 feet:  165 degrees at  52 knots
 6.  Winds at  18000 feet:  210 degrees at  25 knots
```

```
<< RUN SECOND PROGRAM:  S U R F A C E   D R I F T     >>
/\/\/\/\/\/\/   N E W   S C R E E N   \/\/\/\/\/\/\


(***************************************************************************)
(*          SEARCH PLANNING SOFTWARE (PROGRAM #2 OF 3)                     *)
(*  TITLE:         SURFDRIF.COM (Surface Drift Algorithm)                  *)
(*  VERSION:       1.1  for CP/M Operating System                          *)
(*  DATE WRITTEN:  September  1984                                         *)
(*  LICENSE:       COPYRIGHT 1984     D. RICK DOUGLAS                      *)
(***************************************************************************)
The  author  makes no express or implied warranty of any kind with  regard  to
this program material,  including, but not limited to, the implied warranty of
fitness  for  a  particular  purpose.  The  author  shall  not  be  liable  for
incidental  or  consequential  damages in connection with or  arising  out  of
furnishing, use, or performance of this program.  The reader MUST HAVE a solid
understanding  of search and rescue methodology before using this software  in
making  decisions where human life is at risk.   In fact,  since no amount  of
testing  can uncover 100% of program errors,  this program is recommended  for
training  use  only.   Prior  attendance at the United  States  Coast  Guard's
National SAR School is highly-encouraged.

(************************** WARNING! ***************************************)
(*      THIS  SOFTWARE  MAY  BE  FREELY-DISTRIBUTED  PROVIDED  NO  FEE      *)
(*         IS  CHARGED  AND  THIS   COPYRIGHT  NOTICE  IS  RETAINED         *)
(***************************************************************************)

PLEASE HIT RETURN (Once or twice, as necessary) TO CONTINUE


/\/\/\/\/\/\/   N E W   S C R E E N   \/\/\/\/\/\/\


Given the initial surface position coordinates (latitude and
longitude) of an object on the ocean's surface and starting
date/time, this program calculates an updated surface position
(Datum point) or search area for a specified (Datum) time.  If
the updated surface position coordinates are all ready known,
then enter N to the next question and proceed with other recovery
planning as outlined in the National Search and Rescue Manual.


P.S.  Run this program TWICE if the number of hours of search
object drift is uncertain.  On the first run, enter data only
for the SHORTER drift period (later drift start time).  After
the program calculates the Average Surface Wind and Wind Current
vectors, record them, ABORT the program run, and CLEAR your
microcomputer's memory.  Then rerun the entire program, making
sure to enter data for the LONGER drift period (earlier drift
start time).  Enter the vectors you recorded above when asked
by the program.
```

```
Do you wish to continue with this program?  (y/n) => Y

   /\/\/\/\/\/\/    N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\

Please enter the number for the month the search object was
at the last known position:

 1 = Jan        4 = Apr        7 = Jul        10 = Oct
 2 = Feb        5 = May        8 = Aug        11 = Nov
 3 = Mar        6 = Jun        9 = Sep        12 = Dec

LAST KNOWN POSITION MONTH=> 4

Please enter the year the search object was at the last known
position  (e.g., 1985, 1986, etc.)
LAST KNOWN POSITION YEAR=> 1984

   /\/\/\/\/\/\/    N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\

Please enter the day-hour-minute (TO THE NEAREST HOUR) the
search object was at the last known position.  Enter it in
Z DTG (Zulu Date-Time-Group) format.  For example: 9:37 PM,
August 4, should appear as 042200 Greenwich-Mean Time (Z=0).

If the number of hours of search object drift is uncertain,
you must run this program twice (as mentioned earlier).  On
the first run enter here the time the SHORTER drift period
started (LATER than the last known position time).  On the
second run, or if two runs are not applicable, enter here the
the LKP time as requested above:
LAST KNOWN POSITION TIME (Z DTG)=> 291000

   /\/\/\/\/\/\/    N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\

Was search object's last known latitude north or south?
(Enter N or S)      Answer=> N

Please enter the search object's last known latitude
(For example: 25-Degrees 45-Minutes 13-Seconds = 25.4513)
LATITUDE => 39.1848

Was search object's last known longitude east or west?
(Enter E or W)      Answer=> W
```

Please enter the search object's last known longitude
(For example: 160-Degrees 45-Minutes 13-Seconds = 160.4513)
LONGITUDE => 64.3345

/\/\/\/\/\/\/ N E W S C R E E N \/\/\/\/\/\/\

Now you will be asked to enter the year, month, and date/time for the desired Datum position, which MUST be later than the time you just entered for the last known position!

Please enter the desired Datum month:

1 = Jan	4 = Apr	7 = Jul	10 = Oct
2 = Feb	5 = May	8 = Aug	11 = Nov
3 = Mar	6 = Jun	9 = Sep	12 = Dec

DATUM MONTH=> 4

Please enter the desired Datum year (e.g., 1985, 1986, etc.)
DATUM YEAR=> 1984

Please enter the desired Datum day-hour-minute (TO THE NEAREST HOUR).
Enter it in Z DTG (Zulu Date-Time-Group) format. For example:
9:37 PM, August 4, should appear as 042200 Greenwich-Mean Time (Z=0).
DATUM TIME (Z DTG)=> 291800

/\/\/\/\/\/\/ N E W S C R E E N \/\/\/\/\/\/\

This program includes wind current calculations to determine Datum. However, according to the National SAR Manual, wind currents are usually ignored in coastal, lake, river, and harbor areas due to the many variable effects from the water-land interface. This program is based on the assumption of open-sea search where land masses do not interfere with the action of the wind on the water or on the currents generated by them. A rule of thumb is to calculate wind currents when water depths are greater than 100 feet (32 meters) and at distances of 20 miles (32 kilometers) or greater from shore. Wind currents are not usually used inside these limits except where local knowledge makes it possible to estimate one. This is especially true where, close to shore, the water's depth increases rapidly.

```
Do you wish to continue with this program?  (y/n) => Y

   /\/\/\/\/\/\/    N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\

Please enter the number of hours elapsed (to the NEAREST
HOUR) from the last known surface position (LKP) time to
the desired Datum time.


If the number of hours of search object drift is uncertain,
you must run this program twice (as mentioned earlier).  On
the first run enter here the SHORTER number of hours of
search object drift.  On the second run, or if two runs are
not applicable, enter here the difference between the Datum
and LKP times as requested above:
ELAPSED HOURS => 8

   /\/\/\/\/\/\/    N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\

Please enter the ocean surface wind direction and velocity at  271800Z:
WIND DIRECTION = 80
WIND VELOCITY = 20

Please enter the ocean surface wind direction and velocity at  280000Z:
WIND DIRECTION = 40
WIND VELOCITY = 30

Please enter the ocean surface wind direction and velocity at  280600Z:
WIND DIRECTION = 35
WIND VELOCITY = 40

Please enter the ocean surface wind direction and velocity at  281200Z:
WIND DIRECTION = 40
WIND VELOCITY = 40

Please enter the ocean surface wind direction and velocity at  28 1800Z:
WIND DIRECTION = 45
WIND VELOCITY = 20

Please enter the ocean surface wind direction and velocity at  290000Z:
WIND DIRECTION = 40
WIND VELOCITY = 25

Please enter the ocean surface wind direction and velocity at  290600Z:
WIND DIRECTION = 65
WIND VELOCITY = 20
```

```
Please enter the ocean surface wind direction and velocity at  291200Z:
WIND DIRECTION = 74
WIND VELOCITY = 20

Please enter the ocean surface wind direction and velocity at  291800Z:
WIND DIRECTION = 77
WIND VELOCITY = 18

   /\/\/\/\/\/\/     N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\

Refer to the latitude coefficient-vectors table for the Northern
latitudes (approximately page 8-16c) in the National SAR Manual.
Find the degrees-latitude column nearest to 39-degrees-North.
You will now be asked to enter the directions and magnitudes for
each of the eight periods appearing under that column.

PERIOD #1 DIRECTION = 217
PERIOD #1 MAGNITUDE = .024

PERIOD #2 DIRECTION = 350
PERIOD #2 MAGNITUDE = .01

PERIOD #3 DIRECTION = 107
PERIOD #3 MAGNITUDE = .008

PERIOD #4 DIRECTION = 223
PERIOD #4 MAGNITUDE = .006

PERIOD #5 DIRECTION = 339
PERIOD #5 MAGNITUDE = .006

PERIOD #6 DIRECTION = 95
PERIOD #6 MAGNITUDE = .005

PERIOD #7 DIRECTION = 211
PERIOD #7 MAGNITUDE = .004

PERIOD #8 DIRECTION = 327
PERIOD #8 MAGNITUDE = .004
```

```
   /\/\/\/\/\/\/     N  E  W    S  C  R  E  E  N     \/\/\/\/\/\/\

Wind Current Latitude Coefficients for 39-degrees North are:

     1.     Period #1    217 /  0.024
     2.     Period #2    350 /  0.010
     3.     Period #3    107 /  0.008
     4.     Period #4    223 /  0.006
     5.     Period #5    339 /  0.006
     6.     Period #6    95 /  0.005
     7.     Period #7    211 /  0.004
     8.     Period #8    327 /  0.004

Are these altitudes, wind directions, and wind
velocities all correct? (y/n) Y

   /\/\/\/\/\/\/     N  E  W    S  C  R  E  E  N     \/\/\/\/\/\/\

Period #1
Xcomponent =  -0.119  (-0.119; 0% ERROR)
Ycomponent =  0.229   ( 0.229; 0% ERROR)

Period #2
Xcomponent =  -0.283  (-0.283; 0% ERROR)
Ycomponent =  0.109   ( 0.109; 0% ERROR)

Total Wind Current:
Xcomponent =  -1.443  (-1.454; 0.69% ERROR)
Ycomponent =  1.470   ( 1.466; 0.27% ERROR)

HIT RETURN (Once or twice, as necessary) TO CONTINUE

   /\/\/\/\/\/\/     N  E  W    S  C  R  E  E  N     \/\/\/\/\/\/\

Total Wind Current Direction is  316-degrees for  2.06 nautical miles.

If the number of hours of search object drift is uncertain, you
must run this program twice (as mentioned earlier).  On the first
run, RECORD this Total Wind Current vector direction and distance
for input during the second run.

If this is the second run, please now ENTER the previously-
recorded Wind Current vector over the shorter drift period
(If not applicable, enter a heading of 361):
WIND CURRENT DIRECTION (from Run #1) => 361
```

/\/\/\/\/\/\/ N E W S C R E E N \/\/\/\/\/\/\

Average Surface Wind:
Xcomponent = 148.742 (148.742; 0%)
Ycomponent = 39.711 (39.711; 0%)

Average Surface Wind Direction 75-degrees True at 19.24 knots.

If the number of hours of search object drift is uncertain, you must run this program twice (as mentioned earlier). On the first run, RECORD this Average Surface Wind vector direction and speed for use in calculating the minimum leeway speed during the second program run. Then, ABORT this program run, CLEAR your microcomputer's memory, and RERUN the entire program

HIT RETURN (Once or twice, as necessary) TO CONTINUE

/\/\/\/\/\/\/ N E W S C R E E N \/\/\/\/\/\/\

This program will now calculate the LEEWAY vector, or, that drift caused by average surface winds pushing on the exposed area of the search object. There are three leeway vector calculation options:

(1) DRIFT RATE UNCERTAINTY - Used when the search object is unknown, or, when it is not known whether the object has deployed an anti-drift device (e.g., sea drogue). All drift is computed as a minimum and maximum distance downwind;

(2) DRIFT TIME UNCERTAINTY - Used if the number of hours the search object has been drifting is not known for certain. All drift is computed as a minimum and maximum distance downwind;

(3) DIRECTIONAL DRIFT UNCERTAINTY - Used if above options do not apply. However, drift is computed for a divergent bearing left and right of the downwind vector.

Refer to the National SAR Manual for additional information.

PLEASE SELECT OPTION #1, #2, or #3 => 3

/\/\/\/\/\/\/ N E W S C R E E N \/\/\/\/\/\/\

Refer to the Leeway Speed Graph in the National SAR Manual (approximately page 8-15). Please enter the search object's maximum expected degrees-divergence from the downwind vector:
MAX EXPECTED DIVERGENCE => 35

Refer to the Leeway Speed Formulae in the National SAR Manual (approximately page 8-13). Please enter the leeway speed at which the search object drifts:
LEEWAY SPEED => 0.842

Please enter the number of hours the search object drifted:
HOURS OF DRIFT => 8

/\/\/\/\/\/\/ N E W S C R E E N \/\/\/\/\/\/\

Please enter the sea current vector as described in the National SAR Manual (approximately pages 8-16i and 8-16j):
SEA CURRENT DIRECTION (SET) => 71

SEA CURRENT VELOCITY => 1.1

Which publication did you use as the source of your sea current data?:

(1) Naval Oceanographic Office Spec. Pub. Series 4000, Surface Currents
(2) Publication No. 700
(3) Oceanographic Atlas
(4) Atlas of Surface Currents
(5) Pilot Charts
(6) Other

SELECT ONLY ONE => 1

/\/\/\/\/\/\/ N E W S C R E E N \/\/\/\/\/\/\

Please enter the observed total water current, if known. This current is determined by observing the drift positions and times of surface debris, oil slicks, or, by inserting an electronic Datum Marker Buoy in the search area. This total water current vector replaces the surface wind and sea current vectors previously calculated by this program. (If unknown, enter a heading of 361):
TOTAL WATER CURRENT DIRECTION => 361

```
  /\/\/\/\/\/\/    N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\

If you have calculated a Tidal Current vector, please enter it here
(If not applicable, enter a heading of 361):
TIDAL CURRENT DIRECTION => 361

  /\/\/\/\/\/\/    N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\

(Max) Datum Vector:
Xcomponent =  0.550
Ycomponent =  6.645


(Min) Datum Vector:
Xcomponent =  2.543
Ycomponent =  -0.821


Total Surface Drift Dir. :    108-degrees       5-degrees
Total Surface Drift Dist.:    2.67 naut. mi.    6.67 naut. mi.




HIT RETURN (Once or twice, as necessary) TO CONTINUE

  /\/\/\/\/\/\/    N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\

Calculating . . . Please stand by

  /\/\/\/\/\/\/    N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\

A record of significant input and output data used during this
program run is stored in an external file named "SEADATA".
If you desire to keep this record permanently, please rename
file SEADATA before running this program again!
```

<< S E C O N D P R O G R A M O U T P U T >>

/\/\/\/\/\/\/ N E W S C R E E N \/\/\/\/\/\/\

Missing aerospace object/pilot(s) last known position: 39-18 North 64- 33 West
Time: 291000Z 4/1984

==

AVERAGE SURFACE WINDS:

Date-Time-Group	Hours	Wind	Contributions
291200Z	5	74 / 20	74 / 100
291800Z	3	77 / 18	77 / 54

Total Average Surface Wind Direction 075-degrees True at 19.24 knots

==

LEEWAY DRIFT VECTOR CALCULATIONS: Directional Drift Uncertainty

Maximum Expected Divergence	35-degrees	
Drift Rate	0.842 knots	
Hours Of Drift	8	
Leeway Direction	Min = 220-degrees	Max = 290-degrees True
Leeway Distance	Min = 6.74 naut.mi.	Max = 6.74 naut.mi.

==

WIND CURRENT PERIOD #1

Interval	Date-Time-Group	Wind	Coefficients	Contributions
1	291200Z	74 / 20	217 / 0.024	291 / 0.480
2	290600Z	65 / 20	350 / 0.010	415 / 0.200
3	290000Z	40 / 25	107 / 0.008	147 / 0.200
4	281800Z	45 / 20	223 / 0.006	268 / 0.120
5	281200Z	40 / 40	339 / 0.006	379 / 0.240
6	280600Z	35 / 40	95 / 0.005	130 / 0.200
7	280000Z	40 / 30	211 / 0.004	251 / 0.120
8	271800Z	80 / 20	327 / 0.004	407 / 0.080

Local Wind Current This Period	333 / 0.258
Number Of Hours In This Period	5
Wind Current Vector This Period	333 / 1.289 nautical miles

WIND CURRENT PERIOD #2

Interval	Date-Time-Group	Wind	Coefficients	Contributions
1	291800Z	77 / 18	217 / 0.024	294 / 0.432
2	291200Z	74 / 20	350 / 0.010	424 / 0.200
3	290600Z	65 / 20	107 / 0.008	172 / 0.160
4	290000Z	40 / 25	223 / 0.006	263 / 0.150
5	281800Z	45 / 20	339 / 0.006	384 / 0.120
6	281200Z	40 / 40	95 / 0.005	135 / 0.200
7	280600Z	35 / 40	211 / 0.004	246 / 0.160
8	280000Z	40 / 30	327 / 0.004	367 / 0.120

```
Local Wind Current This Period            291 / 0.303
Number Of Hours In This Period            3
Wind Current Vector This Period           291 / 0.909 nautical miles
-----------------------------------------------------------------------
Total Wind Current Direction     316-degrees True
Total Wind Current Distance      2.06 nautical miles
=======================================================================

SEA CURRENT VECTOR:
Used Naval Oceanographic Office Spec. Pub. Series 4000, Surface Currents
Sea Current Direction (Set)      071-degrees True
Sea Current Drift Rate           1.10 knots
Sea Current Drift Distance:      8.80 knots
=======================================================================

TOTAL SURFACE DRIFT (Dmin & Dmax) VECTORS:
Tot.Surf.Drift Direction :   108-degrees      005-degrees True
Tot.Surf.Drift Distance  :   2.67 naut. mi.   6.67 naut. mi.
=======================================================================


   <<  RUN  THIRD PROGRAM:  S E A R C H  A R E A DETERMINATION>>
   /\/\/\/\/\/\/     N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\


(***************************************************************************)
(*          SEARCH PLANNING SOFTWARE (PROGRAM #3 OF 3)                    *)
(*  TITLE:         AREA.COM (Search Area Determination Algorithm)         *)
(*  VERSION:       1.0  for CP/M Operating System                         *)
(*  DATE WRITTEN:  August  1984                                           *)
(*  LICENSE:       COPYRIGHT 1984     D. RICK DOUGLAS                     *)
(***************************************************************************)
The  author  makes no express or implied warranty of any kind with  regard  to
this program material,  including, but not limited to, the implied warranty of
fitness  for  a  particular  purpose.  The  author shall  not  be  liable  for
incidental  or  consequential  damages in connection with or  arising  out  of
furnishing, use, or performance of this program.  The reader MUST HAVE a solid
understanding  of search and rescue methodology before using this software  in
making  decisions where human life is at risk.   In fact,  since no amount  of
testing  can uncover 100% of program errors,  this program is recommended  for
training  use  only.   Prior  attendance at the United  States  Coast  Guard's
National SAR School is highly-encouraged.

(****************************** WARNING! ***********************************)
(*      THIS  SOFTWARE  MAY  BE  FREELY-DISTRIBUTED  PROVIDED  NO  FEE     *)
(*         IS  CHARGED  AND  THIS   COPYRIGHT  NOTICE  IS  RETAINED        *)
(***************************************************************************)

PLEASE HIT RETURN (Once or twice, as necessary) TO CONTINUE
```

```
   /\/\/\/\/\/\/    N E W   S C R E E N    \/\/\/\/\/\/\

Was search object's last known latitude north or south?
(Enter N or S)      Answer=> N

Please enter the search object's last known latitude
(For example: 25-Degrees 45-Minutes 13-Seconds = 25.4513)
LATITUDE => 39.15

Was search object's last known longitude east or west?
(Enter E or W)      Answer=> W

Please enter the search object's last known longitude
(For example: 160-Degrees 45-Minutes 13-Seconds = 160.4513)
LONGITUDE => 64.3

   /\/\/\/\/\/\/    N E W   S C R E E N    \/\/\/\/\/\/\

Did you previously calculate an aerospace drift vector
for this problem?  (y/n): Y

Please enter the previously-calculated,
TOTAL AEROSPACE DRIFT DIRECTION => 323

Please enter the previously-calculated
TOTAL AEROSPACE DRIFT DISTANCE => 4.39  (4.39; 0% ERROR)

Refer to the "Individual Drift Error" section in the National SAR
Manual (approximately page 8-27).  Please enter the desired
Aerospace Drift Error Confidence Factor (e.g, 0.125, 0.3, etc.):
AEROSPACE DRIFT ERROR CONFIDENCE FACTOR => 0.3

   /\/\/\/\/\/\/    N E W   S C R E E N    \/\/\/\/\/\/\

Please enter the sum of previous drift errors
(Enter a zero if not applicable):
SUM OF DRIFT ERRORS => 0

Please enter the previously-calculated,
MINIMUM TOTAL SURFACE DRIFT DIRECTION => 109

Please enter the previously-calculated,
MINIMUM TOTAL SURFACE DRIFT DISTANCE => 2.69  (2.67; 0.74% ERROR)

Please enter the previously-calculated,
MAXIMUM TOTAL SURFACE DRIFT DIRECTION => 005
```

Please enter the previously-calculated,
MAXIMUM TOTAL SURFACE DRIFT DISTANCE => 6.63   (6.67; 0.6% ERROR)

Please enter the measured distance between the two, previously-calculated, Total Surface Drift Distance positions:
DISTANCE BETWEEN SURFACE DRIFT POSITIONS => 7.73

Refer to the "Individual Drift Error" section in the National SAR Manual (approximately page 8-27). Please enter the desired Surface Drift Error Confidence Factor (e.g, 0.125, 0.3, etc.):
SURFACE DRIFT ERROR CONFIDENCE FACTOR => 0.3

/\/\/\/\/\/\/   N E W   S C R E E N   \/\/\/\/\/\/\

Refer to the "Initial Position Error" section in the National SAR Manual (approximately page 8-29). Please enter the search object's Navigational Fix Error (e.g, 0, 5, 10, or 15 naut. mi. radius), and, if applicable, the Navigational Dead-Reckoning (DR) Error (e.g, 0, 5, 10, or 15 percent of DR distance in naut. mi.):
SEARCH OBJECT NAVIGATIONAL FIX ERROR => 10

SEARCH OBJECT NAVIGATIONAL DR ERROR => 0

/\/\/\/\/\/\/   N E W   S C R E E N   \/\/\/\/\/\/\

Refer to the "Search Craft Error" section in the National SAR Manual (approximately page 8-29). Please enter the search craft's Navigational Fix Error (e.g, 0, 5, 10, or 15 naut. mi. radius), and, if applicable, the Navigational Dead-Reckoning (DR) Error (e.g, 0, 5, 10, or 15 percent of DR distance in naut. mi.):
SEARCH CRAFT NAVIGATIONAL FIX ERROR => 5

SEARCH CRAFT NAVIGATIONAL DR ERROR => 0

/\/\/\/\/\/\/   N E W   S C R E E N   \/\/\/\/\/\/\

You are calculating the search area for:

1 The first search
2 The second search
3 The third search
4 The fourth search
5 The fifth search

Input SEARCH NUMBER => 1

```
PRELIMINARY INFORMATION:
------------------------
Search Area Radius =    15 naut. mi.
Search Area        =   900 sq-naut.mi.   (900; 0% ERROR)



HIT RETURN (Once or twice, as necessary) TO CONTINUE

  /\/\/\/\/\/\/    N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\

Calculating . . . Please stand by

  /\/\/\/\/\/\/    N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\

A record of significant input and output data used during this
program run is stored in an external file named "AREADATA".
If you desire to keep this record permanently, please rename
file AREADATA before running this program again!

  <<        T H I R D   P R O G R A M   O U T P U T        >>
  /\/\/\/\/\/\/    N  E  W    S  C  R  E  E  N    \/\/\/\/\/\/\

-----------------------------------------------------------------------
DRIFT COORDINATES              :    LATITUDE      LONGITUDE
Last Known Position            :     39.1500       64.3000
Aerospace Drift Position       :     39.1830       64.3325
Surface Dmin Position          :     39.1738       64.3008
Surface Dmax Position          :     39.2507       64.3240

=======================================================================
SEARCH AREA:

Aerospace Drift Distance       :      4.39 naut. mi.
Drift Error Confidence         :      0.300
Aerospace Drift Error          :      1.32 naut. mi.
-----------------------------------------------------------------------
Sum of Previous Drift Errors   :      0.00 naut. mi.
Surface Drift Distance         :      2.69 naut. mi.     6.63 naut. mi.
Drift Error Confidence         :      0.300
Drift Error Min and Max        :      0.81 naut. mi.     1.99 naut. mi.
Distance Between Dmin/max      :      7.73 naut. mi.
Surface Drift Error Minimax    :      5.26 naut. mi.
-----------------------------------------------------------------------
```

Total Drift Error	:	6.58 naut. mi.	
Object Navigation Fix Error	:	10 naut. mi.	
Object Dead-Reckn Error	:	0.00 naut. mi.	
Object Total Position Error	:	10.00 naut. mi.	
Searcher Navigation Fix Error	:	5 naut. mi.	
Searcher Dead-Reckn Error	:	0.00 naut. mi.	
Searcher Total Position Error	:	5.00 naut. mi.	
Total Probable Error	:	12.97 naut. mi.	
Safety Factor	:	1.1	
Search Radius	:	15 naut. mi.	
Search Area	:	900 naut. mi.-squared.	
SEARCH AREA COORDINATES:	:	LATITUDE	LONGITUDE
Center Point	:	39.2414	64.2923
Corner Point (Upper Left)	:	39.3908	64.4834
Corner Point (Lower Left)	:	39.0908	64.4834
Corner Point (Upper Right)	:	39.3908	64.0945
Corner Point (Lower Right)	:	39.0908	64.0945

Conclusion

The only significant hardware calculation errors introduced in the sample USCG problem are those in determining the search area center and corner point coordinates. Inaccuracies here were as large as 6 minutes of latitude (6 nautical miles) and 2 minutes of longitude. However, drift vector error discrepancies were miniscule to non-existent. As such, the attached programs can be used to generate these vectors, which can then be plotted on navigation charts to more accurately determine drift position coordinates during search planning.

Appendix D

SEARCH PLANNING PROGRAMS LISTING (SOURCE CODE)

```
(****************************************************************)
(*                                                              *)
(*        SEARCH PLANNING SOFTWARE (PROGRAM #1 OF 3)            *)
(*                                                              *)
(* TITLE:         AERODRIF.COM (Aerospace Drift Algorithm)      *)
(* VERSION:       1.1  for CP/M Operating System                *)
(* DATE WRITTEN:  September 1984                                *)
(* ------------------------------------------------------------ *)
(* DESCRIPTION:                                                 *)
(*   - User asked for last known position (coordinates) and     *)
(*     date-time-group of search object                         *)
(*   - User queried as to whether aerospace object glided,      *)
(*     fell, parachuted, or some combination, to get to the     *)
(*     surface                                                  *)
(*   - Depending upon the descent method, user is then asked    *)
(*     additional information as specified in the National      *)
(*     Search and Rescue (SAR) Manual                           *)
(*   - User asked to input reported winds aloft for every       *)
(*     known altitude from surface to the search object's       *)
(*     incident level, plus one or two more above that          *)
(*   - Wind data is printed back to the user for verification   *)
(*   - User may make changes to input data as required          *)
(*   - Program calculates applicable, resultant drift vector    *)
(*     direction(s) and magnitude(s) and creates an "audit      *)
(*     trail"/record file of program input, significant         *)
(*     calculations, and output, named "AERODATA"               *)
(* ------------------------------------------------------------ *)
(* LICENSE:  COPYRIGHT 1984     D. RICK DOUGLAS                 *)
(*                                                              *)
(* The author makes no express or implied warranty of any       *)
(* kind with regard to this program material, including, but    *)
(* not limited to, the implied warranty of fitness for a        *)
(* particular purpose.  The author shall not be liable for      *)
(* incidental or consequential damages in connection with or    *)
(* arising out of furnishing, use, or performance of this       *)
(* program.  The reader MUST HAVE a solid understanding of      *)
(* search and rescue methodology before using this software     *)
(* in making decisions where human life is at risk.  In fact,   *)
(* since no amount of testing can uncover 100% of program       *)
(* errors, this program is recommended for training use or [illegible]. *)
(* Prior attendance at the United States Coast Guard's          *)
(* National SAR School is highly-encouraged.                    *)
(*                                                              *)
(*        THIS  SOFTWARE  MAY  BE  FREELY-DISTRIBUTED           *)
(*        PROVIDED  NO  FEE  IS  CHARGED  AND                   *)
(*        THIS  COPYRIGHT  NOTICE  IS  RETAINED.                *)
(* ------------------------------------------------------------ *)
```

```
(*  LANGUAGE: PASCAL                                           *)
(*  USED    : Borland International, TURBO.PAS, Version 2.0    *)
(*  ---------------------------------------------------------- *)
(*  MODULES CALLED (Sequentially listed); (OPT) = "Optional":  *)
(*                                                             *)
(*                     AeroPosition                            *)
(*                     VerifyDTG                               *)
(*  ______________________|______________________              *)
(*  PATH #1             PATH #2             PATH #3            *)
(*  GLIDE ONLY          PARACHUTE ONLY      BOTH METHODS       *)
(*     |                   |                   |               *)
(*  GlideData           ChuteData           GlideData          *)
(*  AeroGlide           InputWinds          AeroGlide          *)
(*  InputWinds          VerifyWinds         InputWinds         *)
(*  VerifyWinds         WindChart           VerifyWinds        *)
(*  WindChart           WindDrift           WindChart          *)
(*  WindDrift           GlideOrChute        WindDrift          *)
(*  GlideOrChute        AddVectors          GlideThenChute     *)
(*  AddVectors          Ejection            AddVectors         *)
(*  AddVectors (OPT)    AddVectors (OPT)    AddVectors (OPT)   *)
(*  WriteToDisk         WriteToDisk         Ejection           *)
(*                                          AddVectors (OPT)   *)
(*                                          WindDrift          *)
(*                                          GlideOrChute       *)
(*                                          AddVectors         *)
(*                                          AddVectors (OPT)   *)
(*                                          WriteToDisk        *)
(*                                                             *)
(***************************************************************)

program AeroSpaceWinds (input,output);

const radians = 57.2957795; {Standard radian conversion factor}

type Winds = array[1..20,1..3] of real;

var OnePassCompleted : boolean;
    {Flags second pass through procedure WindDrift}

    chute,                    {Determines if Drift procedure should be used}
    continue,                 {Determines if this program should be used}
    descent,                  {Determines if AeroGlide procedure should be used}
    LatNS,                    {Indicates whether latitude is North or South}
    LongEW,                   {Indicates whether longitude is East or West}
    MethodDrift   : char;     {Indicates parachute(P), glide(G), or both(B) }

    GlideRatio,               {Horizont./Vert. displacement (from flight manual) }
    I,J,                      {Used as counters}
    N,                        {Tracks number of wind altitudes used by program}
    LastMonth,                {Month object at last known position}
    LastYear,                 {Year object at last known position}
    TypeVehicle   :integer;   {Determines if vehicle is fast or slow}
```

```
     AssignedAlt,           {Incident altitude above mean sea level (MSL)}
     AvgWindFrom,           {Average winds aloft direction from (in degrees) }
     AvgWindTo,             {AvgWindFrom plus-or-minus 180}
     BrgGlideWinds,         {Average winds aloft glide bearing in degrees}
     BrgMaxGlide,           {Maximum glide course in degrees}
     BrgParaDrift,          {Average winds aloft bearing on parachute drift}
     BrgRads,               {Individual bearings converted to radians}
     ChuteAlt,              {Determines parachute(s) opening altitude}
     DescentHeading,        {Aerospace object's descent heading in decimal}
     DescentRate,           {Aerospace object's descent rate in feet per minute}
     DescentTime,           {Object's descent/glide time through GlideAltLost}
     DistanceParaDrift,     {Parachute drift distance}
     EjectDistance,         {Distance ejecting pilot is tossed horizontally}
     GlideAltLost,          {Altitude lost falling from AssignedAlt}
     GlideDistance,         {Glide ratio distance covered excluding winds}
     GlideWindSpeed,        {Average winds aloft velocity affecting glide}
     IncidentAlt,           {Parachute-opening altitude}
     LastLatitudeKnown,     {Last known latitude of aerospace object}
     LastLongitudeKnown,    {Last known longitude of aerospace object}
     LastDateTime,          {Time aerospace object was at last known position}
     MaxDistanceGlide,      {In nautical miles}
     MaxRadiusGlide,        {Maximum radius of glide if BrgMaxGlide uncertain}
     ParaDriftSpeed,        {Average winds aloft parachute drift velocity}
     RateParaDescent,       {Object's parachute descent rate in feet per second}
     ResultMagnitude,       {Resultant distance computed from vector addition}
     Temp,                  {Confirms WindDrift IncidentAlt/500 remainder = 0}
     TerrainHeight,         {Height of surface at or above mean sea level}
     TimeParaDescent,       {Object's parachute descent time in minutes}
     TotalBrg,              {Total aerospace drift vector direction}
     TotalDistance,         {Total aerospace drift distance}
     TotalRadius,           {Total possible area radius if direction unknown}
     TrajectBrg,            {Aerospace trajectory direction}
     TrajectDistance,       {Aerospace trajectory distance}
     TrajectRadius,         {Aerospace trajectory radius (TrajectBrg uncertain)}
     WindDisplacement : real;{Distance covered due to winds}

     No,                        {'N,n' = "No" characters}
     Yes,                       {'Y,y' = "Yes" characters}
     ValidAnswers : set of char; {'Y,y,N,n' = valid "Yes" and "No" characters}

     AeroWinds,             {Duplicate array of WindsAloft for computation use}
     WindsAloft    : Winds; {Individual-altitude winds affecting parachute drift}

{**********************************************************************}
{Writes out a specified number of blank lines.}

procedure writelns (lines : integer);

var count : integer;

begin
```

```
   count := 0;
   while lines ) count do
      begin
         writeln;
         count := count + 1
      end
end;

{**********************************************************************}
{Verifies legitimate date/time/group data input.}

procedure VerifyDTG;

var Temp,                   {Used as a temporary computation variable}
    TempDate,               {Used as temporary date variable}
    TempHour,               {Used as temporary hour variable}
    TempMinutes : real;     {Used as temporary minutes variable}

begin  {VerifyDTG}
   repeat {until correct date/time format input}

      {Verify the day is between 1 and 31}
      TempDate := trunc( LastDateTime/10000 );
      if (TempDate ( 1) or (TempDate ) 31) then
      begin  {TempDate not between 1 and 31}
         writeln;
         writeln('You have incorrectly entered the date.  Try again!');
         write('Re-enter Z DTG=) ');
         readln(LastDateTime)
      end;  {TempDate not between 1 and 31}

      {Verify the hour is between 0000 and 2400}
      Temp := (LastDateTime/10000 - (trunc(LastDateTime/10000)) ) * 100;
      TempHour := trunc(Temp);
      if (TempHour ( 0) or (TempHour ) 23) then
      begin {TempHour not between 0 and 23}
         writeln;
         writeln('You have incorrectly entered the hour.  Try again!');
         write('Re-enter Z DTG=) ');
         readln(LastDateTime)
      end;  {TempHour not between 0 and 23}

      {Verify the minute is between 0 and 60}
      Temp := (LastDateTime/100 - (trunc(LastDateTime/100)) ) * 100;
      TempMinutes := trunc(Temp);
      if (TempMinutes ( 0) or (TempMinutes ) 60) then
      begin  {TempMinutes not between 0 and 60}
         writeln;
         writeln('You have incorrectly entered the minutes.  Try again!');
         write('Re-enter Z DTG=) ');
         readln(LastDateTime)
      end   {TempMinutes not between 0 and 60}
```

```
   until (TempDate >= 1) and (TempDate <= 31) and (TempHour >= 0) and
         (TempHour <= 23) and (TempMinutes >= 0) and (TempMinutes <= 60)

end;  {VerifyDTG}

{**********************************************************************}
{Asks user for last known time and position of an aerospace object.}

procedure AeroPosition;

begin  {AeroPosition}

   writeln;
   repeat  {until valid response}
      writeln('Please enter the number for the month the search object was');
      writeln('at the last known position:');
      writeln;
      writeln(' 1 = Jan        4 = Apr        7 = Jul        10 = Oct');
      writeln(' 2 = Feb        5 = May        8 = Aug        11 = Nov');
      writeln(' 3 = Mar        6 = Jun        9 = Sep        12 = Dec');
      writeln;
      write('MONTH=> ');
      readln(LastMonth);
      if (LastMonth < 1) or (LastMonth > 12) then
         writeln('Incorrect number entered.  Please try again!');
      writeln
   until (LastMonth >= 1) and (LastMonth <= 12);

    writeln('Please enter the year the search object was at the last known');
    writeln('position  (e.g., 1985, 1986, etc.)');
    write('YEAR=> ');
    readln(LastYear);
    writeln;

   writeln;
   writeln('Please enter the day-hour-minute the search object was at the');
   writeln('last known position.  Enter it in Z DTG (Zulu Date-Time-Group)');
   writeln('format.  For example: 9:37 PM, August 4, would appear as');
   writeln('042137 Greenwich-Mean Time (Z=0).');
   write('TIME (Z DTG)=> ');
   readln(LastDateTime);
   VerifyDTG;
   writeln;

   repeat {until valid latitude}
      writeln('Was search object''s last known latitude north or south?');
      write('(Enter N or S)      Answer=> ');
      readln(LatNS);
      writeln
   until (LatNS = 'N') or (LatNS = 'n') or (LatNS = 'S') or (LatNS = 's');
```

```
   repeat {until valid latitude}
     writeln('Please enter the search object''s last known latitude ');
     writeln('(For example: 25-Degrees 45-Minutes 13-Seconds = 25.4513)');
     write('LATITUDE => ');
     readln(LastLatitudeKnown);
     if (LastLatitudeKnown < 0) or (LastLatitudeKnown > 90) then
       writeln('Input latitude must be between 0-90.  Try again!');
     writeln
   until (LastLatitudeKnown >= 0) and (LastLatitudeKnown <= 90);

   repeat {until valid longitude}
     writeln('Was search object''s last known longitude east or west?');
     write('(Enter E or W)      Answer=> ');
     readln(LongEW);
     writeln
   until (LongEW = 'E') or (LongEW = 'e') or (LongEW = 'W') or (LongEW = 'w');

   repeat {until valid longitude}
     writeln('Please enter the search object''s last known longitude ');
     writeln('(For example: 160-Degrees 45-Minutes 13-Seconds = 160.4513)');
     write('LONGITUDE => ');
     readln(LastLongitudeKnown);
     if (LastLongitudeKnown < 0) or (LastLongitudeKnown > 180) then
       writeln('Input longitude must be between 0-180.  Try again!');
     writeln
   until (LastLongitudeKnown >= 0) and (LastLongitudeKnown <= 180);
   writeln

end;  {AeroPosition}

{*************************************************************************}
{When given the X and Ycomponents from procedure VectorComponents, this}
{procedure calculates resultant vector bearing (AvgWindFrom) and       }
{magnitude (ResultMagnitude).                                          }

procedure AddVectors (var Xcomponent, Ycomponent : real);

var TempAngle : real;  {Temporary calculation variable}

begin  {AddVectors}

   {Find resultant angle (in degrees) uncorrected for compass position.}
   TempAngle := arctan(Xcomponent/Ycomponent) * radians;

   {Find bearing resulting from TempAngle corrected for compass position.}
   if (Xcomponent > 0) and (Ycomponent > 0) then AvgWindFrom := TempAngle;
   if ((Xcomponent > 0) and (Ycomponent < 0)) or
     ((Xcomponent < 0) and (Ycomponent < 0)) then
     if (TempAngle + 180) > 360 then
       AvgWindFrom := TempAngle - 180
     else AvgWindFrom := TempAngle + 180;
   if (Xcomponent < 0) and (Ycomponent > 0) then
```

```
      if (TempAngle + 360) ) 360 then
         AvgWindFrom := TempAngle - 360
      else AvgWindFrom := TempAngle + 360;

   {Convert AvgWindFrom to AvgWindTo bearing.}
   if (AvgWindFrom + 180) ) 360 then
      AvgWindTo := AvgWindFrom - 180
   else AvgWindTo := AvgWindFrom + 180;

   {Resultant bearing magnitude is the square root of the sum of}
   { the squared components.}
   ResultMagnitude := sqrt(Xcomponent * Xcomponent  +  Ycomponent * Ycomponent);

   writeln('Xcomponent = ',Xcomponent:5:3);
   writeln('Ycomponent = ',Ycomponent:5:3);
   writeln
end;  {AddVectors}

{**********************************************************************}
{This procedure is used if the aerospace object only glides (or falls), or   }
{parachutes (after, or in lieu of, gliding) to the surface .  It's purpose is}
{to find the altitudes halfway between each altitude for which a wind        }
{direction and velocity have been reported.   It begins with the next higher }
{altitude winds reported above the surface level winds, and ends with that   }
{mid-altitude that is greater than or equal to the altitude at which the     }
{descent began.  It then uses these mid-altitudes to determine the affective }
{wind component velocities on the descending aerospace object.               }

procedure GlideOrChute (var IncidentAlt: real);

var I          : integer; {Used as counters}
    Temp1,                {Used in wind component velocity calculations}
    Temp2     : real;     {Used in wind component velocity calculations}
    LowAltitude,          {Determines lower range altitude for wind component}
    HighAltitude          {Determines upper range altitude for wind component}
               : array[1..20] of real;

begin  {GlideOrChute}
   {Initialize variables}
   for I := 1 to 20 do HighAltitude[I] := 0.0;  Temp1 := 0.0;
   for I := 1 to 20 do LowAltitude[I] := 0.0;   Temp2 := 0.0;

   {Calculate wind components for each applicable altitude.}
   {If IncidentAlt = 500ft. then halve the wind velocity component.}
   if IncidentAlt = 500.0 then AeroWinds[1,3] := AeroWinds[1,3] * 0.5;
   {If IncidentAlt = 1000ft., then wind velocity component remains unchanged.}

   if N ) 1 then  {Confirms parachute opened at or above 1500 feet}
   begin  {N ) 1 loop}
      HighAltitude[1] := 1000.0;

      {Start calculations for altitudes above 1000-ft}
```

```
      I := 2;
      repeat

         {HighAltitude = altitude halfway between present one and next}
         {higher one, unless HighAltitude is greater than IncidentAlt.}
         HighAltitude[I] := (AeroWinds[I+1,1] - AeroWinds[I,1])
                                                   /2 + AeroWinds[I,1];
         {LowAltitude = previous HighAltitude.}
         LowAltitude[I] := HighAltitude[I-1];

         if IncidentAlt >= HighAltitude[I] then
         begin  {IncidentAlt >= HighAltitude[I]}
            Temp1 := (HighAltitude[I] - LowAltitude[I]) / 1000.0;
            AeroWinds[I,3] := AeroWinds[I,3] * Temp1;
            I := I + 1;

            if IncidentAlt = HighAltitude[I] then
            begin {IncidentAlt = HighAltitude[I]}
               {Make next two altitude's wind components = 0}
               AeroWinds[I+1,3] := 0.0; AeroWinds[I+2,3] := 0.0;
               I := N  {Exit I = 2 to N loop}
            end   {IncidentAlt = HighAltitude[I]}
         end   {IncidentAlt >= HighAltitude[I]}
         else
         begin  {IncidentAlt < HighAltitude[I]}
            Temp2 := (IncidentAlt - LowAltitude[I]) / 1000.0;
            AeroWinds[I,3] := AeroWinds[I,3] * Temp2;

            {Make next two altitude's wind components = 0}
            AeroWinds[I+1,3] := 0.0; AeroWinds[I+2,3] := 0.0;
            I := N  {Exit I = 2 to N loop}
         end  {IncidentAlt < HighAltitude[I]}
      until I = N;

      write('Wind Data (Glide or Parachute Opening Altitude to');
      writeln(' surface level AGL)');
      writeln('With Velocity Components Calculated Prior To Vector Addition:');
      writeln;
      for I := N downto 1 do
      begin  {for I = 1 to number of individual wind vectors}
         write('  Winds at ',AeroWinds[I,1]:6:0,' feet: ');
         write(AeroWinds[I,2]:4:0,' degrees at ',AeroWinds[I,3]:3:0);
         writeln(' knots')
      end;  {for I = 1 to number of individual wind vectors}
      writeln
   end   {N > 1 loop}
end;  {GlideOrChute}

{*************************************************************************}
{This procedure is used if the aerospace object glides (or falls) to a lower }
{altitude at which parachute(s) open to slow it's descent to the surface.    }
{It's purpose is to find the altitudes halfway between each altitude for     }
```

```
{which a wind direction and velocity have been reported.  It begins with the }
{mid-altitude that is lower than or equal to the parachute(s) opening        }
{altitude, and ends with the mid-altitude that is higher than or equal to the}
{altitude at which the descent began.  It then uses these mid-altitudes to   }
{determine affective wind component velocities on the gliding (or falling)  }
{portion of the aerospace object's descent.                                  }

procedure GlideThenChute (var IncidentAlt: real);

var I,H,L     : integer; {Used as counters}
    Temp1,               {Used in wind component velocity calculations}
    Temp2     : real;    {Used in wind component velocity calculations}
    LowAltitude,         {Determines lower range altitude for wind component}
    HighAltitude         {Determines upper range altitude for wind component}
              : array[1..20] of real;

begin  {GlideThenChute}
   {Initialize variables}
   for I := 1 to 20 do HighAltitude[I] := 0.0;  Temp1 := 0.0;
   for I := 1 to 20 do LowAltitude[I] := 0.0;   Temp2 := 0.0;

   {Searches for where the ChuteAlt fits in between AeroWinds array}
   {altitudes, starting at surface level.}
   I := 1;  {Surface level}

   repeat
      if ChuteAlt > AeroWinds[I,1] then I := I + 1
   until ChuteAlt <= AeroWinds[I,1];

   {Determines midpoints between AeroWinds array wind altitudes.     }
   {Then inserts ChuteAlt and IncidentAlt to calculate wind component}
   {for each reported altitude's actual velocity range of altitudes  }
   {between midpoints.                                               }

   {First, find the midpoint between AeroWinds array altitudes       }
   {between which ChuteAlt is located.}
   LowAltitude[I] := ( (AeroWinds[I,1]-AeroWinds[I-1,1]) / 2 ) +
                                                     AeroWinds[I-1,1];

   {Next, find the midpoint between the next given AeroWinds array   }
   {altitude (above ChuteAlt) and the next altitude above that one.  }
   HighAltitude[I] := ( (AeroWinds[I+1,1]-AeroWinds[I,1]) / 2 ) +
                                                        AeroWinds[I,1];

   {If ChuteAlt is at or above the midpoint (LowAltitude), but below }
   {the next given AeroWinds array altitude, then the velocity       }
   {component = the given altitude - ChuteAlt all divided by 1000.   }
   if ChuteAlt >= LowAltitude[I] then
   begin
      N := I;  {Initialize N at AeroWinds array altitude above midpoint}
      Temp1 := (AeroWinds[I,1] - ChuteAlt) / 1000
   end
```

```
else
{If ChuteAlt is below the midpoint (LowAltitude), but above the   }
{last given AeroWinds array altitude, then the velocity component }
{equals the midpoint - ChuteAlt all divided by 1000.              }
begin
   N := I-1;   {Initialize N at AeroWinds array altitude below midpoint}
   Temp1 := 0; {Ensures Temp1 has no effect on below calculations}
   AeroWinds[I-1,3] := ( (LowAltitude[I] - ChuteAlt) / 1000) *
                                                          AeroWinds[I-1,3]
end;
L := N;

{If IncidentAlt is at or above the midpoint between the AeroWinds }
{array altitude and the next higher one, then the velocity        }
{component equals the midpoint - the given altitude, all divided  }
{by 1000.                                                         }
if IncidentAlt >= HighAltitude[I] then
Temp2 := (HighAltitude[I] - AeroWinds[I,1]) / 1000
else
{If IncidentAlt is below the midpoint between the AeroWinds array }
{altitude and the next higher one, then the velocity component    }
{equals the IncidentAlt - the given altitude all divided by 1000. }
Temp2 := (IncidentAlt - AeroWinds[I,1]) / 1000;

{Total wind component effect equals the sum of component effects  }
{above and below the midpoint between AeroWinds array wind         }
{altitudes times the given altitudes velocity.                     }
AeroWinds[I,3] := (Temp1 + Temp2) * AeroWinds[I,3];

repeat  {until valid response}
   if IncidentAlt > HighAltitude[I] then
   begin  {IncidentAlt > HighAltitude[I]}
      I := I + 1;
      {HighAltitude = altitude halfway between present one and next}
      {higher one, unless HighAltitude is greater than IncidentAlt.}
      HighAltitude[I] := (AeroWinds[I+1,1] - AeroWinds[I,1])
                                                /2 + AeroWinds[I,1];
      if IncidentAlt < HighAltitude[I] then HighAltitude[I] := IncidentAlt;

      {LowAltitude = previous HighAltitude.}
      LowAltitude[I] := HighAltitude[I-1];

      if IncidentAlt > HighAltitude[I] then
      begin  {IncidentAlt >= to present altitude limit.}
         Temp1 := (HighAltitude[I] - LowAltitude[I]) / 1000.0;
         AeroWinds[I,3] := AeroWinds[I,3] * Temp1
      end   {IncidentAlt >= to present altitude limit.}
      else
      begin  {IncidentAlt < to present altitude limit.}
         Temp2 := (IncidentAlt - LowAltitude[I]) / 1000.0;
         AeroWinds[I,3] := AeroWinds[I,3] * Temp2;
         H := I;
```

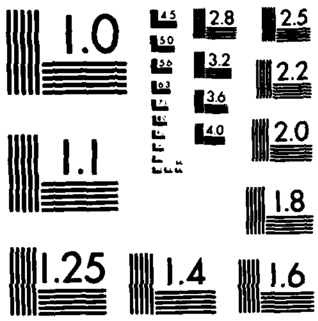
```

            {Zero out final altitude/wind components if next-to-last}
            {altitude = IncidentAlt.}
            if AeroWinds[N-1,3] = IncidentAlt then AeroWinds[I+1,3] := 0
          end    {IncidentAlt < to present altitude limit.}
        end   {IncidentAlt > HighAltitude[I]}
      until IncidentAlt <= HighAltitude[I];

      writeln('Wind Data (Glide to Parachute Opening Altitude AGL) With');
      writeln('Velocity Components Calculated Prior To Vector Addition:');
      writeln;
      for I := H downto L do
      begin  {for I = number of individual wind vectors}
        write('  Winds at ',AeroWinds[I,1]:6:0,' feet: ');
        write(AeroWinds[I,2]:4:0,' degrees at ',AeroWinds[I,3]:3:0);
        writeln(' knots')
      end;  {for I = number of individual wind vectors}
      writeln

    end;   {GlideThenChute}

    {++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
    {If aerospace object(s) of interest is a (are) pilot(s) or astronaut(s)      }
    {ejecting from a crippled vehicle, this procedure asks the  user for         }
    {(judgmental) performance-type of vehicle.  Then it sets the ejection        }
    {distance equal to the standards for that type of vehicle as prescribed in   }
    {the National SAR Manual.                                                    }

    procedure Ejection;

    begin  {Ejection}

      repeat  {until valid response}
        writeln('If aerospace object is a pilot ejecting from a');
        writeln('crippled aerospace vehicle, was that vehicle:');
        writeln;
        writeln('1.  A turboprop or medium performance jet aircraft');
        writeln('2.  A high-performance jet aircraft');
        writeln('3.  Neither of the above');
        writeln;
        write('ENTER A 1, 2, or 3 => ');
        readln(TypeVehicle);
        writeln
      until (TypeVehicle > 0) or (TypeVehicle < 4);

      if TypeVehicle = 1 then EjectDistance := 0.5;
      if TypeVehicle = 2 then EjectDistance := 0.8;
      if TypeVehicle = 3 then EjectDistance := 0.0

    end;   {Ejection}

    {++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
```

```
{Determines the resultant drift vector bearing and magnitude.}

procedure WindDrift (var IncidentAlt: real);

var I,J         : integer; {Used as counters}
    BrgRads,               {Individual bearings converted to radians}
    TempAlt,               {Used for interim IncidentAlt calculations}
    Xcomponent,            {Horizontal component of individual vectors}
    Ycomponent  : real;    {Vertical component of individual vectors}

begin  {WindDrift}
   {Make duplicate array of WindsAloft for computation purposes}
   for I := 1 to N do for J := 1 to 3 do AeroWinds[I,J] := WindsAloft[I,J];

   {Call procedure GlideOrChute if aerospace object uses no parachute(s) to  }
   {descend to surface level.                                                }
   if ( (AssignedAlt-GlideAltLost) = TerrainHeight) and (MethodDrift = 'G') then
      GlideOrChute(IncidentAlt);

   {Call procedure GlideOrChute if aerospace object uses only parachute(s) to}
   {descend to surface level (no glide or free-fall involved).              }
   if MethodDrift = 'P' then GlideOrChute(IncidentAlt);

   {Ensures the following instructions are used only for the program's      }
   {initial pass through procedure WindDrift.                               }
   if OnePassCompleted = false then

      {If aerospace object glides or falls to a lower altitude before using  }
      {parachutes to descend to surface level, then call procedure           }
      {GlideThenChute for the "fall or glide portion" of the descent.        }
      if (AssignedAlt - GlideAltLost) >= 500.0 then GlideThenChute(IncidentAlt);

   {At this point, individual wind-component-contributions exist in the}
   {cells of AeroWinds array; We now calculate the resultant magnitude }
   {and velocity of these vectors using procedure AddVectors.          }
   Xcomponent := 0.0;   Ycomponent := 0.0;       {re-initialize}
   for I := 1 to (N - 1) do
   begin  {calculate Xcomponent and Ycomponent sums}
      {Convert all vector's individual bearings into radian units.}
      BrgRads := AeroWinds[I,2] / radians;

      {Total all AeroWinds individual X and Y vector components.}
      Xcomponent := Xcomponent + sin(BrgRads) * AeroWinds[I,3];
      Ycomponent := Ycomponent + cos(BrgRads) * AeroWinds[I,3]
   end;  {calculate Xcomponent and Ycomponent sums}

   AddVectors(Xcomponent,Ycomponent);

   if MethodDrift  = 'G' then
   begin  {Gliding or falling object}
      BrgGlideWinds := AvgWindTo;
      TempAlt := GlideAltLost / 1000;
```

AD-A151 833 MICROCOMPUTER APPLICATION OF AEROSPACE ASSET SURFACE SEARCH PLANNING(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING 2/2

UNCLASSIFIED D R DOUGLAS 14 DEC 84 AFIT/GSO/MATH/84D-1 F/G 6/7 NL

END



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
      GlideWindSpeed := ResultMagnitude /TempAlt;
   end;  {Gliding or falling object}

   if MethodDrift = 'P' then
   begin  {Aerospace object uses parachute(s) descent}
      BrgParaDrift := AvgWindTo;
      TempAlt := IncidentAlt / 1000;
      ParaDriftSpeed := ResultMagnitude /TempAlt;
      writeln('The average winds aloft affecting the parachute drift of');
      write('the aerospace object are bearing to ',BrgParaDrift:3:0);
      writeln(' degrees at ',ParaDriftSpeed:3:1,' knots.');
      writeln;
      writeln('Refer to the "Parachute Drift Table" in the National');
      writeln('Search and Rescue Manual (approximately page 8-12) and');
      writeln('interpolate parachute drift distance (in nautical miles)');
      writeln('from the published chart.  For example: For a parachute');
      writeln('opening altitude of 3000 feet (915 meters) and winds aloft');
      writeln('averaging 15 knots, the parachute drift distance');
      writeln('interpolates to 0.675 nautical miles).');
      write('PARACHUTE DRIFT DISTANCE => ');
      readln(DistanceParaDrift);
      writeln;

      writeln('Refer to the "Parachute Descent Data Table" in the National');
      writeln('Search and Rescue Manual (approximately page 8-11) and');
      writeln('determine the parachute descent rate (in feet-per-second)');
      writeln('from the published chart.  For example: A 28-foot (C-9)');
      writeln('escape parachute at 7000 feet lowers a person at a rate of');
      writeln('21.4 ft/sec, while the three, 83-ft.-diameter parachutes');
      writeln('on the Apollo space capsule allowed for a 32.5 ft/sec');
      writeln('descent at this same altitude.');
      write('PARACHUTE DESCENT RATE => ');
      readln(RateParaDescent);
      writeln;

      {Calculate parachute descent rate in minutes}
      TimeParaDescent := IncidentAlt / (RateParaDescent*60);

      if OnePassCompleted = false then
      begin  {procedure AeroGlide has not been used by program}
         repeat  {until valid compass heading}
            writeln('Please enter descent or bailout heading, if known');
            write('(Enter 361 if not known).  HEADING (0-361) => ');
            readln(DescentHeading);
            if (DescentHeading < 0) or (DescentHeading > 361) then
            writeln('This heading must be between 0-361.  Try again!');
            writeln
         until (DescentHeading >= 0) and (DescentHeading <= 361);
         Ejection;

         {Finds maximum aerospace displacement vector}
         BrgRads := BrgParaDrift / radians;
```

```
            Xcomponent := sin(BrgRads) * DistanceParaDrift;
            Ycomponent := cos(BrgRads) * DistanceParaDrift;

            if (DescentHeading < 361) and (EjectDistance > 0.0) then
            begin  {descent heading known and EjectDistance significant}
               BrgRads := DescentHeading / radians;
               Xcomponent := Xcomponent + sin(BrgRads) * EjectDistance;
               Ycomponent := Ycomponent + cos(BrgRads) * EjectDistance;
               AddVectors(Xcomponent,Ycomponent);
               TotalBrg := AvgWindFrom;
               TotalDistance := ResultMagnitude
            end   {descent heading known and EjectDistance significant}
            else
            begin  {descent heading and EjectDistance not both significant}
               TotalBrg := BrgParaDrift;
               TotalDistance := DistanceParaDrift;
               if DescentHeading = 361 then
                  if EjectDistance > 0.0 then
                     TotalRadius := EjectDistance
            end   {descent heading and EjectDistance not both significant}
         end;  {procedure AeroGlide has not been used by program}

         if OnePassCompleted = true then
         begin  {Adding parachute data to glide descent data for total vector.}
            {Finds total aerospace displacement vector}
            BrgRads := BrgParaDrift / radians;
            Xcomponent := sin(BrgRads) * DistanceParaDrift;
            Ycomponent := cos(BrgRads) * DistanceParaDrift;
            BrgRads := TrajectBrg / radians;
            Xcomponent := Xcomponent + sin(BrgRads) * TrajectDistance;
            Ycomponent := Ycomponent + cos(BrgRads) * TrajectDistance;
            AddVectors(Xcomponent,Ycomponent);
            TotalBrg := AvgWindFrom;
            TotalDistance := ResultMagnitude;
            TotalRadius := TrajectRadius
         end   {Adding parachute data to glide descent data for total vector.}
      end;  {Aerospace object uses parachute(s) descent}

      OnePassCompleted := true
   end;  {WindDrift}

   {************************************************************************}
   {Prints keyboard-input wind altitudes, directions, and velocities, on the}
   {video screen for user verification.}

   procedure WindChart;

   var I : integer;  {Used as a counter}

   begin  {WindChart}
      for I := 1 to N do
      begin  {for I = 1 to number of individual wind vectors}
```

```
      write(I:2,'.  Winds at ',WindsAloft[I,1]:6:0,' feet: ');
      write(WindsAloft[I,2]:4:0,' degrees at ',WindsAloft[I,3]:3:0);
      writeln(' knots')
    end   {for I = 1 to number of individual wind vectors}
end;  {WindChart}

{*****************************************************************}
{Allows user to change wind altitudes, directions, and velocities, entered}
{below in procedure InputWinds.}

procedure VerifyWinds;

var WindCheck : char;     {Used to verify input wind data}
    J         : integer;  {Used as a counter}
    WindError : integer;  {Used to indicate which input wind data is in error}

begin  {VerifyWinds}
   WindCheck := 'Y';  WindError := -1;  {initialize}

{Verify input wind directions and velocities.}
   WindChart;  {Prints current wind data on video screen}

   repeat  {until WindCheck = valid response}
      writeln;
      writeln('Are these altitudes, wind directions, and wind');
      write('velocities all correct? (y/n) ');
      readln(WindCheck);
      writeln
   until (WindCheck in ValidAnswers);

   while (WindCheck in No) do
   begin  {while WindCheck = NO}
      repeat  {until WindError = 0 to 20}
         writeln('Which line is in error?');
         write('Enter number, or zero for none) => ');
         readln(WindError);
         writeln
      until (WindError >= 0.0) and (WindError < 21.0);

      if WindError > 0.0 then
      begin  {if an input line is identified as being in error}
         write('Please enter altitude, wind direction ');
         writeln('and velocity to replace:');
         write(WindsAloft[WindError,1]:4:0,' ');
         write(WindsAloft[WindError,2]:4:0);
         write(' ',WindsAloft[WindError,3]:4:0,' => ');
         for J := 1 to 3 do read(WindsAloft[WindError,J]);
         writelns(2)
      end;   {if an input line is identified as being in error}

      WindChart;  {Prints current wind data on video screen}

```

```
      writeln;
      repeat  {until valid WindCheck response}
         write('Are all other lines correct? (y/n) ');
         readln(WindCheck);
         writeln
      until (WindCheck in ValidAnswers)

   end;  {while WindCheck = NO}
   CLRSCR
end;   {VerifyWinds}

{##########################################################################}
{Queries user for altitudes at which the aerospace-object begins to fall or  }
{glide, or to deploy parachute(s).  Then it asks for wind directions and     }
{velocities at those altitudes through which the object descends.  The user  }
{is allowed to verify/change any input data before it is used in determining }
{the resultant drift vector bearing and magnitude by procedure ParaDrift.    }

procedure InputWinds (var IncidentAlt : real);

var I,                    {Used as counter}
    P          : integer; {Tracks # of input wind altitudes above IncidentAlt}
    Temp2      : real;    {Confirms input wind altitudes are multiples of 1000}

begin  {InputWinds}
   P := 0;  {Initialize}
   CLRSCR;

   {Starts bulk of winds aloft computations if IncidentAlt >= 500ft.}
   if IncidentAlt >= 500.0 then
   begin {winds aloft computations}

      {Begin entry of individual wind components for applicable altitudes.}
      writeln;
      writeln('PLEASE ENTER REPORTED OR FORECAST WIND DIRECTION AND VELOCITY');
      writeln('FOR EACH KNOWN ALTITUDE WITHIN THE FOLLOWING CONSTRAINTS:');
      writeln;
      writeln('1. You must input the altitude, wind direction (0 to 360), and');
      writeln('   velocity (0 to 99), from surface level up to the altitude at');
      writeln('   which the power-off descent began or parachute(s) opened,');
      writeln('   whichever is higher.  You will then be asked to enter one');
      writeln('   higher altitude for which wind data is known.  Finally, you');
      writeln('   will be able to verify your input data before any');
      writeln('   computations are made.');
      writeln;
      writeln('2. Altitudes must be entered in rounded thousands of feet,');
      writeln('   e.g., 3000, 7000, 12000, not 3500, 5200, etc.');
      writeln;
      writeln('3. An example follows of input altitude/winds data format:');
      writeln;
      write('   ALTITUDE      =  0 (Surface level); Enter lowest');
      writeln(' altitude first!');
```

```
writeln('   WIND DIRECTION = 330');
writeln('   WIND VELOCITY  = 25');
writeln;
write('HIT RETURN (Once or twice, as necessary) TO CONTINUE');
readln(continue);  {Pauses program before next clear-screen command}
CLRSCR;

{Input surface winds}
write('Begin altitude/wind data input starting with surface');
writeln(' level winds:');
writeln('ALTITUDE       = 0');

repeat {until wind direction = 0-360 degrees}
   write('WIND DIRECTION = ');
   readln(WindsAloft[1,2]);
   if (WindsAloft[1,2] < 0.0) or (WindsAloft[1,2] > 360.0) then
   begin  {Wind direction not in 0-360 degree range}
      writeln('Wind direction must be between 0 and 360. Try again!');
      writeln
   end    {Wind direction not in 0-360 degree range}
until (WindsAloft[1,2] >= 0.0) and (WindsAloft[1,2] <= 360.0);

repeat {until wind velocity = 0-99 (knots, or whatever) }
   write('WIND VELOCITY  = ');
   readln(WindsAloft[1,3]);
   if (WindsAloft[1,3] < 0.0) or (WindsAloft[1,3] > 99.0) then
      writeln('Wind velocity must be between 0 and 99. Try again!');
   writeln
until (WindsAloft[1,3] >= 0.0) and (WindsAloft[1,3] <= 99.0);

{Input winds at other altitudes}
I := 1;  {Used to repeat loop until all valid winds are input}
N := 1;  {Used to note number of altitudes used}

if (IncidentAlt - TerrainHeight) >= 1500 then
begin  {for altitudes >= 1500 feet}

   repeat  {until I = 21}
      I := I + 1; {I = 2 to 20 individual altitudes/winds data input loop}
      N := N + 1; {Notes number of altitudes used}

      {Repeat altitude input process until the next altitude input is at}
      {least 1000 feet higher than the previous one, greater than 1000, }
      {1000 feet, and less than or equal to 42000 feet.}
      repeat
         write('ALTITUDE       = ');
         readln(WindsAloft[I,1]);

         if WindsAloft[I,1] < WindsAloft[I-1,1] then
         begin   {Input altitude must be greater than previous one input}
            write('This altitude must be higher than the last altitude');
            writeln(' input.  Try again!');
```

```
            writeln
         end;    {Input altitude must be greater than previous one input}

         if WindsAloft[I,1] < 1000.0 then
         begin  {Input altitude must be >= than 1000}
            writeln('This altitude must be >= 1000. Try again!');
            writeln
         end;   {Input altitude must be >= than 1000}

         Temp2 :=(WindsAloft[I,1] /1000.0)-trunc(WindsAloft[I,1] /1000.0);
         if Temp2 <> 0.0 then
         begin  {WindsAloft must be a multiple of 1000}
            write('This altitude must be a multiple of 1000 feet.');
            writeln('  Try again!');
            writeln
         end;   {WindsAloft must be a multiple of 1000}

         if WindsAloft[I,1] > 42000.0 then
         begin  {Input altitude must be less than 42000}
            writeln('This altitude must be <= 42000. Try again!');
            writeln
         end    {Input altitude must be less than 42000}

      until (WindsAloft[I,1] > WindsAloft[I-1,1]) and
            (WindsAloft[I,1] >= 1000.0) and (WindsAloft[I,1] <= 42000.0);

      repeat {until wind direction = 0-360 degrees}
         write('WIND DIRECTION = ');
         readln(WindsAloft[I,2]);
         if (WindsAloft[I,2] < 0.0) or (WindsAloft[I,2] > 360.0) then
         begin  {Wind direction not in 0-360 degree range}
            write('Wind direction must be between 0 and 360. Try again!');
            writeln;
            writeln
         end    {Wind direction not in 0-360 degree range}
      until (WindsAloft[I,2] >= 0.0) and (WindsAloft[I,2] <= 360.0);

      repeat {until wind velocity = 0-99 (knots, or whatever) }
         write('WIND VELOCITY  = ');
         readln(WindsAloft[I,3]);
         if (WindsAloft[I,3] < 0.0) or (WindsAloft[I,3] > 99.0) then
            writeln('Wind velocity must be between 0 and 99. Try again!');
         writeln
      until (WindsAloft[I,3] >= 0.0) and (WindsAloft[I,3] <= 99.0);

      {P notes first altitude >= IncidentAlt (AGL) and requests one more}
      if (WindsAloft[I,1] >= IncidentAlt) then
      begin  {Input altitude >= IncidentAlt (AGL)}
         P := P + 1;
         if P = 1 then
            writeln('Enter data for just one more, higher altitude.');
         writeln
```

```
            end;  {Input altitude )= IncidentAlt (AGL)}

      {P notes second altitude}IncidentAlt (AGL) then terminates the input loop}
         {by setting I equal to 21.}
            if (WindsAloft[I,1] )= IncidentAlt) and (P = 2) then I := 21

         until I = 21
      end;  {for altitudes )= 1500 feet}

      CLRSCR;
      VerifyWinds;
      WindDrift(IncidentAlt)

   end   {winds aloft computations}

end; {InputWinds}

{***********************************************************************}
{Asks user for aerospace object glide information, if applicable.}

procedure AeroGlide;

const FtNm = 6076;

var Xcomponent,          {Horizontal component of individual vectors}
    Ycomponent  : real;  {Vertical component of individual vectors}

begin  {AeroGlide}
   Xcomponent := 0.0;  Ycomponent := 0.0;  {initialize}

   repeat  {until valid Glide Ratio}
      writeln('Please enter the aerospace object''s power-off glide ratio.');
      writeln('(Refer to object''s operations manual, if one exists.)');
      writeln('Otherwise, enter best estimate.  For example:  If the object');
      writeln('glides 3 feet horizontally for every foot of vertical descent');
      write('(3/1), enter a 3.    GLIDE RATIO =) ');
      readln(GlideRatio);
      if GlideRatio ( 0 then
         writeln('Glide Ratio must not be less than zero.  Try again!');
      writeln
   until GlideRatio )= 0;

   GlideDistance := (GlideRatio * GlideAltLost) / FtNm;   {Converts ft to nm}

   repeat  {until valid compass heading}
      writeln('Please enter descent or bailout heading, if known');
      write('(Enter 361 if not known).  HEADING (0-361) =) ');
      readln(DescentHeading);
      if (DescentHeading ( 0) or (DescentHeading ) 361) then
         writeln('This heading must be between 0-361.  Try again!');
      writeln
   until (DescentHeading )= 0) and (DescentHeading (= 361);
```

```
   repeat  {until valid descent rate}
      writeln('Please enter aerospace object''s rate of descent in');
      write('feet-per-minute.  DESCENT RATE => ');
      readln(DescentRate);
      if DescentRate < 0 then
         writeln('Descent rate must not be less than zero.  Try again!');
      writeln
   until DescentRate >= 0;
   DescentTime := GlideAltLost / DescentRate;

{Returns average wind affecting object from incident altitude to parachute}
{opening altitude or surface level.}
   InputWinds(AssignedAlt);
   WindDisplacement := (GlideWindSpeed / 60) * DescentTime;

   if DescentHeading < 361 then  {Descent or bailout heading known}
   begin  {Calculate resultant vector of Glide + Wind vectors.}

      {Convert winds bearing on gliding object into radian units.}
      BrgRads := BrgGlideWinds / radians;
      Xcomponent := sin(BrgRads) * WindDisplacement;
      Ycomponent := cos(BrgRads) * WindDisplacement;

      BrgRads := DescentHeading / radians;
      Xcomponent := Xcomponent + sin(BrgRads) * GlideDistance;
      Ycomponent := Ycomponent + cos(BrgRads) * GlideDistance;
      AddVectors(Xcomponent,Ycomponent);
      BrgMaxGlide := AvgWindTo;
      MaxDistanceGlide := ResultMagnitude
   end   {Calculate resultant vector of Glide + Wind vectors.}
   else
   begin  {descent or bailout heading unknown}
      BrgMaxGlide := BrgGlideWinds;
      MaxDistanceGlide := WindDisplacement;
      MaxRadiusGlide := GlideDistance
   end;  {descent or bailout heading unknown}

   if (AssignedAlt - GlideAltLost) >= 500.0 then
   begin  {Parachute(s) were used}
      Ejection;

      {Calculate resultant vector of MaxGlide + Bailout vectors.}
      BrgRads := BrgMaxGlide / radians;  {Convert bearing into radian units.}
      Xcomponent := sin(BrgRads) * MaxDistanceGlide;
      Ycomponent := cos(BrgRads) * MaxDistanceGlide;

      if (DescentHeading < 361) and (EjectDistance > 0.0) then
      begin  {Descent heading known AND EjectDistance > 0}
         {Convert bearing into radian units.}
         BrgRads := DescentHeading / radians;

```

```
          {Total all vector's individual X and Y components.}
          Xcomponent := Xcomponent + sin(BrgRads) * EjectDistance;
          Ycomponent := Ycomponent + cos(BrgRads) * EjectDistance;
          AddVectors(Xcomponent,Ycomponent);
          TrajectBrg := AvgWindTo;
          TrajectDistance := ResultMagnitude
       end   {Descent heading known AND EjectDistance > 0}
       else
       begin  {Descent heading unknown OR EjectDistance = 0}
          TrajectBrg := BrgMaxGlide;
          TrajectDistance := MaxDistanceGlide;
          if DescentHeading = 361 then
             if EjectDistance > 0.0 then
                TrajectRadius := MaxRadiusGlide + EjectDistance
             else TrajectRadius := MaxRadiusGlide
       end;  {Descent heading unknown OR EjectDistance = 0}

       MethodDrift := 'P';  {Switches from "G" to "P" mode as new MethodDrift}
       WindDrift(ChuteAlt)
    end    {Parachute(s) were used}
end;  {AeroGlide}

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
{Gathers additional information concerning the aerospace object's glide  }
{before commencing calculations.                                          }

procedure GlideData;

begin  {GlideData}
   MethodDrift := 'G';  {Selects "Glide" mode as the method of drift.}

   repeat {Until valid AssignedAlt response}
      write('Approximate (to the nearest 500 feet) at what altitude ');
      writeln('above');
      write('mean sea level (MSL) did the aerospace object''s ');
      writeln('power-off');
      write('descent begin?  (If unknown, enter the MSL height of the ');
      writeln('last');
      write('assigned flight altitude.)   ALTITUDE=> ');
      readln(AssignedAlt);

      {Used to ensure AssignedAlt is a multiple of 500}
      Temp := ( AssignedAlt/500.0 ) - trunc( AssignedAlt/500.0 );
      if Temp <> 0 then
         writeln('Given altitude is not a multiple of 500 ft.');
      writeln
   until AssignedAlt > TerrainHeight;

   AssignedAlt := AssignedAlt - TerrainHeight;  {Convert to AGL altitude}

   repeat  {until valid altitude}
      writeln('Please enter the altitude lost (to the nearest 500 feet)');
```

```
      writeln('between the above altitude and the bailout/parachute');
      writeln('deployment altitude.  Altitude lost must be at least 500');
      writeln('feet.  (If parachutes were not used, enter the altitude');
      writeln('lost before surface contact.)');
      write('ALTITUDE LOST => ');
      readln(GlideAltLost);

      {Used to ensure GlideAltLost is a multiple of 500}
      Temp := ( GlideAltLost/500.0 ) - trunc( GlideAltLost/500.0 );

      if Temp <> 0 then
      begin  {GlideAltLost not multiple of 500 feet}
         write('Given altitude lost is not a multiple of 500 ft.  ');
         writeln('Try again!')
      end;   {GlideAltLost not multiple of 500 feet}

      if GlideAltLost < 500 then
      begin  {GlideAltLost not >= 500 feet}
         write('This altitude must be at least 500 feet.  ');
         writeln('Try again!')
      end;   {GlideAltLost not >= 500 feet}

      if GlideAltLost > AssignedAlt then
      begin  {GlideAltLost greater than possible}
         write('It is impossible to lose more altitude than you have. ');
         writeln('Try again!')
      end;   {GlideAltLost greater than possible}

      writeln
   until (Temp = 0.0) and (GlideAltLost >= 500) and
             (GlideAltLost <= AssignedAlt);

   ChuteAlt := AssignedAlt - GlideAltLost;
   AeroGlide

end;   {GlideData}

{**************************************************************************}
{Gathers additional information concerning the aerospace object's parachute }
{descent before commencing calculations.                                   }

procedure ChuteData;

begin  {ChuteData}

   repeat  {until valid response}
      writeln('After receiving report of the search object''s last known');
      writeln('coordinates (lat/long), did the object successfully use');
      write('parachutes to descend?  (y/n)  => ');
      readln(chute);
      writeln
   until (chute in ValidAnswers);
```

```
   if (chute in Yes) then
   begin  {parachute(s) used}
      MethodDrift := 'P';  {Selects "Parachute" mode as the method of drift.}

      repeat {Until valid IncidentAlt response}
         write('Approximate (to the nearest 500 feet) at what ');
         writeln('altitude');
         write('above mean sea level (MSL) did the parachute(s) ');
         writeln('open?');
         writeln('(Enter a zero if descent began below 500 feet AGL)');
         write('ALTITUDE=) ');
         readln(IncidentAlt);

         if ( (IncidentAlt-TerrainHeight) ( 500 ) and
            (IncidentAlt () 0.0) then
            begin  {IncidentAlt not )= 500 feet AGL}
               write('This altitude must be at least 500 feet AGL.  ');
               writeln('Try again!');
               writeln('(Type a zero if not applicable)')
            end;  {IncidentAlt not )= 500 feet AGL}

            if IncidentAlt = 0 then
            begin  {Parachute(s) not effectively used}
               writeln;
               writeln('By entering a zero to this question you have');
               writeln('indicated that parachute(s) were not used 500');
               writeln('feet or more above surface level.  This means');
               writeln('parachute drift does not significantly affect');
               writeln('the surface position of your search object.');
               writeln('You should, therefore, proceed with search');
               writeln('planning using the object''s last known position');
               writeln('as the surface starting (DATUM) point.');
            end;  {Parachute(s) not effectively used}

            {Used to ensure IncidentAlt is a multiple of 500}
            Temp := ( IncidentAlt/500.0 ) - trunc( IncidentAlt/500.0 );
            writeln
      {IncidentAlt must be 500-42000 feet and in units of 500 feet, or = 0}
      until (IncidentAlt )= 0.0) and (IncidentAlt (= 42000.0) and
               (Temp = 0.0);

      ChuteAlt := IncidentAlt - TerrainHeight;  {Convert to AGL altitude}
      {If ChuteAlt )= 500 feet then continue with program run, else terminate.}
      if ChuteAlt () 0.0 then InputWinds(ChuteAlt)

   end   {parachute(s) used}
end;  {ChuteData}

{######################################################################}
{Prints out record of search planning inputs and calculations from}
{this program.                                                    }
```

```
procedure WriteToDisk;

var I           : integer;
    LastDegree,
    LastMinute  : real;
    AeroDat     : text;

begin  {WriteToDisk}
   assign(AeroDat,'AeroData');
   rewrite(AeroDat);

   LastDegree := trunc(LastLatitudeKnown);
   LastMinute := round( (LastLatitudeKnown-LastDegree) * 100);
   write(AeroDat,'Missing aerospace object/pilot(s) last known position:');
   if LastDegree < 10 then write(AeroDat,' 0',LastDegree:1:0,'-')
   else write(AeroDat,' ',LastDegree:2:0,'-');
   if LastMinute < 10 then write(AeroDat,'0',LastMinute:1:0,' ')
   else write(AeroDat,LastMinute:2:0,' ');
   if LatNS = 'N' then write(AeroDat,'North') else write(AeroDat,'South');

   LastDegree := trunc(LastLongitudeKnown);
   LastMinute := round( (LastLongitudeKnown-LastDegree) * 100);
   if LastDegree < 10 then write(AeroDat,' 0',LastDegree:1:0,'-')
   else write(AeroDat,'  ',LastDegree:2:0,'-');
   if LastMinute < 10 then write(AeroDat,'0',LastMinute:1:0,' ')
   else write(AeroDat,LastMinute:2:0,' ');
   if LongEW = 'W' then writeln(AeroDat,'West') else writeln(AeroDat,'East');

   {Prints a zero in front of one-digit dates in Z DTG format}
   if trunc(LastDateTime / 10000) < 10 then
   writeln(AeroDat,'Time: 0',LastDateTime:5:0,'Z ',LastMonth:2,'/',LastYear:4)
   else
   writeln(AeroDat,'Time: ',LastDateTime:6:0,'Z ',LastMonth:2,'/',LastYear:4);

   if GlideAltLost > 0.0 then
   begin  {indicates procedure AeroGlide was used}
      write(AeroDat,'Incident or last assigned altitude    = ');
      writeln(AeroDat,(AssignedAlt+TerrainHeight):6:0,' feet MSL');
      write(AeroDat,'Surface level or terrain height       = ');
      writeln(AeroDat,TerrainHeight:6:0,' feet MSL');
      write(AeroDat,'Altitude lost during glide or descent = ');
      writeln(AeroDat,GlideAltLost:6:0,' feet');
      write(AeroDat,'Power-off glide ratio (Hz/Vertical)   = ');
      writeln(AeroDat,GlideRatio:6,' to 1');
      write(AeroDat,'Maximum horizontal glide distance     = ');
      writeln(AeroDat,GlideDistance:6:1,' nautical miles');
      if DescentHeading < 361 then
      begin  {descent heading known}
         write(AeroDat,'Descent or bailout heading           = ');
         writeln(AeroDat,DescentHeading:6:0,' degrees True')
      end   {descent heading known}
```

```
      else
      begin  {descent heading unknown}
        write(AeroDat,'Descent or bailout heading          = ');
        writeln(AeroDat,'Unknown')
      end;   {descent heading unknown}
      if DescentRate > 0 then
      begin
        write(AeroDat,'Rate of descent                     = ');
        writeln(AeroDat,DescentRate:6:0,' feet per minute');
        write(AeroDat,'Time of descent                     = ');
        writeln(AeroDat,DescentTime:6:0,' minutes')
      end;
      write(AeroDat,'Average descent wind bearing          = ');
      writeln(AeroDat,BrgGlideWinds:6:0,' degrees True');
      write(AeroDat,'Average descent wind velocity (knots) = ');
      writeln(AeroDat,GlideWindSpeed:6:0,' knots');
      write(AeroDat,'Aircraft displacement due to winds    = ');
      writeln(AeroDat,WindDisplacement:6:1,' nautical miles');
      write(AeroDat,'Maximum glide bearing                 =  ');
      writeln(AeroDat,BrgMaxGlide:6:0,' degrees True');
      write(AeroDat,'Maximum aircraft glide                = ');
      writeln(AeroDat,MaxDistanceGlide:6:1,' nautical miles');
      if MaxRadiusGlide > 0.0 then
      begin  {MaxRadiusGlide> 0 due to uncertain data inputs}
        write(AeroDat,'Maximum glide radius                 = ');
        writeln(AeroDat,MaxRadiusGlide:6:1,' nautical miles.')
      end;   {MaxRadiusGlide> 0 due to uncertain data inputs}
      if EjectDistance > 0.0 then
      begin  {aerospace object is/are pilot(s) ejecting}
        write(AeroDat,'Ejection displacement                = ');
        writeln(AeroDat,EjectDistance:6:1,' nautical miles')
      end;   {aerospace object is/are pilot(s) ejecting}
      write(AeroDat,'Aerospace-trajectory drift bearing    =  ');
      writeln(AeroDat,TrajectBrg:6:0,' degrees True');
      write(AeroDat,'Aerospace-trajectory drift distance   = ');
      writeln(AeroDat,TrajectDistance:6:1,' nautical miles');
      if TrajectRadius > 0.0 then
      begin  {TrajectRadius > 0 due to uncertain data inputs}
        write(AeroDat,'Aerospace-trajectory drift radius    = ');
        writeln(AeroDat,TrajectRadius:6:1,' nautical miles.')
      end    {TrajectRadius > 0 due to uncertain data inputs}
    end;   {indicates procedure AeroGlide was used};

    if MethodDrift = 'P' then
    begin  {indicates parachute(s) used}
      write(AeroDat,'Bailout/Parachute opening altitude    = ');
      writeln(AeroDat,(ChuteAlt+TerrainHeight):6:0,' feet MSL');
      if GlideAltLost = 0.0 then
      begin  {if TerrainHeight not previously recorded via glide calculations}
        write(AeroDat,'Surface level or terrain height       = ');
        writeln(AeroDat,TerrainHeight:6:0,' feet MSL')
      end;   {if TerrainHeight not previously recorded via glide calculations}
```

```
      write(AeroDat,'Wind bearing affecting para-descent   = ');
      writeln(AeroDat,BrgParaDrift:6:0,' degrees True');
      write(AeroDat,'Wind velocity affecting para-descent  = ');
      writeln(AeroDat,ParaDriftSpeed:6:1,' knots');
      write(AeroDat,'Parachute table descent rate          = ');
      writeln(AeroDat,RateParaDescent:6:1,' feet per second');
      write(AeroDat,'Parachute drift table distance        = ');
      writeln(AeroDat,DistanceParaDrift:6:1,' nautical miles');
      if (GlideAltLost = 0.0) and (EjectDistance > 0.0) then
      begin  {if EjectDistance not previously recorded}
         write(AeroDat,'Ejection displacement                 = ');
         writeln(AeroDat,EjectDistance:6:1,' nautical miles')
      end;   {if EjectDistance not previously recorded}
      write(AeroDat,'Total aerospace drift bearing         = ');
      writeln(AeroDat,TotalBrg:6:0,' degrees True');
      write(AeroDat,'Total aerospace drift distance        = ');
      writeln(AeroDat,TotalDistance:6:1,' nautical miles');
      if TotalRadius > 0.0 then
      begin  {TotalRadius > 0 due to uncertain data inputs}
         write(AeroDat,'Total aerospace drift radius          = ');
         writeln(AeroDat,TotalRadius:6:1,' nautical miles')
      end;   {TotalRadius > 0 due to uncertain data inputs}
      write(AeroDat,'Total time of parachute descent       = ');
      writeln(AeroDat,TimeParaDescent:6:1,' minutes');

   end;   {indicates parachute(s) used}

   writeln(AeroDat,' ');
   writeln(AeroDat,'  Wind Data Used To Calculate Above Results:');
   {Write out record of winds aloft used by program}
   for I := 1 to N do
   begin  {I = 1 to number of winds/altitudes used by program}
      write(AeroDat,I:2,'.  Winds at ',WindsAloft[I,1]:6:0,' feet: ');
      write(AeroDat,WindsAloft[I,2]:4:0,' degrees at ',WindsAloft[I,3]:3:0);
      writeln(AeroDat,' knots')
   end;   {I = 1 to number of winds/altitudes used by program}

   close(AeroDat)

end;   {WriteToDisk}

{****************************************************************************}
{Provides user with program information, limitations on use and license.    }

procedure Warranty;

var continue : char;  {Bogus read variable provides time to read warranty}

begin  {Warranty}

   write('(***************************************');
   writeln('**************************************)');
```

```
   write('(*          SEARCH PLANNING SOFTWARE ');
   writeln('(PROGRAM #1 OF 3)                  *)');
   write('(*  TITLE:         AERODRIF.COM (Aerospace Drift ');
   writeln('Algorithm)                *)');
   write('(*  VERSION:       1.1  for CP/M ');
   writeln('Operating System                          *)');
   write('(*  DATE WRITTEN:  September  1984');
   writeln('                                          *)');
   write('(*  LICENSE:       COPYRIGHT 1984');
   writeln('     D. RICK DOUGLAS                      *)');
   write('(*************************************');
   writeln('*************************************)');
   write('The  author  makes no express or implied ');
   writeln('warranty of any kind with  regard  to');
   write('this program material,  including, but ');
   writeln('not limited to, the implied warranty of');
   write('fitness  for  a  particular  purpose.  ');
   writeln('The  author shall  not  be  liable  for');
   write('incidental  or  consequential  damages ');
   writeln('in connection with or  arising  out  of');
   write('furnishing, use, or performance of this ');
   writeln('program.  The reader MUST HAVE a solid');
   write('understanding  of search and rescue ');
   writeln('methodology before using this software  in');
   write('making  decisions where human life is at ');
   writeln('risk.   In fact,  since no amount  of');
   write('testing  can uncover 100% of program ');
   writeln('errors,  this program is recommended  for');
   write('training  use  only.   Prior  attendance ');
   writeln('at the United  States  Coast  Guard''s');
   writeln('National SAR School is highly-encouraged.');
   writeln;
   write('(****************************');
   writeln('  WARNING!  **********************************)');
   write('(*       THIS  SOFTWARE  MAY  BE  FREELY-');
   writeln('DISTRIBUTED  PROVIDED  NO  FEE     *)');
   write('(*          IS  CHARGED  AND  THIS');
   writeln('   COPYRIGHT  NOTICE  IS  RETAINED        *)');
   write('(*************************************');
   writeln('*************************************)');
   writeln;
   write('PLEASE HIT RETURN (Once or twice, as necessary) TO CONTINUE');
   readln(continue)

end;  {Warranty}

{**************************************************************************}
begin  {main program}

   {initialize program variables}
   AssignedAlt := -1000.0;  AvgWindFrom := 0.0;  BrgGlideWinds := 0.0;
   BrgMaxGlide := 0.0;  BrgParaDrift := 0.0;  ChuteAlt := 0.0;
```

```
chute := '0';  descent := '0';  DescentHeading := -1.0;
DescentTime := 0.0;  DistanceParaDrift := 0.0;
EjectDistance := -1.0;  GlideAltLost := 0.0;  GlideRatio := 0;
GlideWindSpeed := 0.0;  IncidentAlt := -1000.0;  LastDateTime := 0.0;
LastLatitudeKnown := 0.0;  LastLongitudeKnown := 0.0;  LatNS := '0';
LongEW := '0';  N := 0;  MaxDistanceGlide := 0.0;  MaxRadiusGlide := 0.0;
MethodDrift := '0';  OnePassCompleted := false;  ParaDriftSpeed := 0.0;
ResultMagnitude := 0.0;  TerrainHeight := -1000.0;  TimeParaDescent := 0.0;
TotalBrg := 0.0;  TotalDistance := 0.0;  TotalRadius := 0.0;
TrajectBrg := 0.0;  TrajectDistance := 0.0;  TrajectRadius := 0.0;
WindDisplacement := 0.0;

ValidAnswers := ['Y','y','N','n'];  No := ['N','n'];  Yes := ['Y','y'];
for I := 1 to 20 do for J := 1 to 3 do WindsAloft[I,J] := 0.0;
for I := 1 to 20 do for J := 1 to 3 do AeroWinds[I,J] := 0.0;

Warranty;
CLRSCR;
writeln('This program calculates the initial surface position coordinates');
writeln('(latitude/longitude) of a falling, gliding, or parachuting');
writeln('aerospace object or person.  If these coordinates are all ready');
writeln('known, or the search object did not fall at least 500 feet, then');
writeln('enter N to the next question and proceed with other search');
writeln('planning as outlined in the National Search and Rescue Manual.');
writeln;
repeat  {until valid response}
   write('Do you wish to continue with this program?  (y/n) => ');
   readln(continue);
   writeln
until (continue in ValidAnswers);
if (continue in Yes) then
begin  {program run}
   CLRSCR;
   AeroPosition;

   repeat  {until valid TerrainHeight response}
      writeln('Please enter the surface level altitude or terrain height');
      writeln('(in feet) above or below mean sea level.  Enter a zero if');
      writeln('altitude equals sea level, or, enter a negative altitude if');
      writeln('below sea level  (e.g. Death Valley, California.).');
      write('SURFACE LEVEL/TERRAIN HEIGHT => ');
      readln(TerrainHeight);
      writeln
   until  TerrainHeight > -1000;

   repeat  {until valid response}
      writeln('After receiving report of the search object''s last known');
      writeln('coordinates (lat/long), did the object fall or glide to a');
      writeln('lower altitude before parachute(s) opened or ground contact');
      write('was made?  (y/n)  => ');
      readln(descent);
      writeln
```

```
        until (descent in ValidAnswers);

        if (descent in Yes) then  GlideData;
        if (descent in No)  then  ChuteData;

        {Only write transaction record to disk if program procedures AeroGlide}
        {or InputWinds were actually used}

        {If program used, then create record of inputs and results.}
        if (MethodDrift = 'P') or (MethodDrift = 'G') or (MethodDrift = 'B') then
          WriteToDisk;
        writeln;
        writeln('A record of significant input and output data used during this');
        writeln('program run is stored in an external file named "AERODATA".');
        writeln('If you desire to keep this record permanently, please rename');
        writeln('file AERODATA before running this program again!')

    end    {program run}
end.  {main program}
```

Variable	Program Line Number									
AddVectors	291	614	680	703	1011	1039				
AeroDat	1198	1201	1202	1206	1207	1208	1209	1210	1211	1211
	1215	1216	1217	1218	1219	1219	1223	1225	1229	1230
	1231	1232	1233	1234	1235	1236	1237	1238	1241	1242
	1246	1247	1251	1252	1253	1254	1256	1257	1258	1259
	1260	1261	1262	1263	1264	1265	1268	1269	1273	1274
	1276	1277	1278	1279	1282	1283	1289	1290	1293	1294
	1296	1297	1298	1299	1300	1301	1302	1303	1306	1307
	1309	1310	1311	1312	1315	1316	1318	1319	1323	1324
	1328	1329	1330	1332						
AeroGlide	952	1121								
AeroPosition	213	1432								
AeroSpaceWinds	78									
AeroWinds	141	352	352	365	365	366	373	373	379	379
	386	386	389	389	400	401	401	438	439	448
	448	449	453	453	454	462	471	472	481	486
	491	491	499	499	500	509	509	514	514	519
	519	529	530	530	580	607	610	611	1413	
AssignedAlt	101	584	598	997	1022	1075	1078	1078	1082	1084
	1084	1110	1118	1120	1230	1397				
AvgWindFrom	102	301	305	306	309	310	313	314	315	681
	704	1397								
AvgWindTo	103	314	315	618	625	1012	1040			
BrgGlideWinds	104	618	1004	1017	1257	1397				
BrgMaxGlide	105	1012	1017	1027	1045	1263	1398			
BrgParaDrift	106	625	629	671	686	697	1297	1398		
BrgRads	107	573	607	610	611	671	672	673	677	678
	679	697	698	699	700	701	702	1004	1005	1006
	1008	1009	1010	1027	1028	1029	1034	1037	1038	
chute	87	1137	1139	1141	1399					
ChuteAlt	108	438	439	459	462	471	1054	1120	1182	1184
	1184	1290	1398							

ChuteData	1129 1454
continue	88 832 1341 1389 1426 1428 1429
count	149 152 153 156 156
descent	89 1399 1449 1451 1453 1454
DescentHeading	109 663 664 664 667 667 675 677 688 979 988 980 983 983 1000 1008 1031 1034 1047 1239 1242 1399
DescentRate	110 988 989 992 993 1249 1252
DescentTime	111 993 998 1254 1400
DistanceParaDrift	112 640 672 673 687 698 699 1303 1400
EjectDistance	113 561 562 563 675 678 679 689 690 1031 1037 1038 1048 1049 1271 1274 1304 1307 1401
Ejection	544 668 1024
FtNm	954 974
GlideAltLost	114 584 598 619 974 993 1022 1093 1096 1096 1104 1110 1117 1118 1120 1227 1234 1291 1304 1401
GlideData	1062 1453
GlideDistance	115 974 1009 1010 1019 1238
GlideOrChute	336 585 589
GlideRatio	94 968 969 972 974 1236 1401
GlideThenChute	419 598
GlideWindSpeed	116 620 998 1259 1402
H	421 515 527
HighAltitude	342 347 357 365 368 370 372 376 425 430 453 480 481 494 499 501 501 504 506 508 522

I	95	338	347	347	348	348	360	365	365	365
	366	368	368	370	372	372	373	373	374	374
	376	379	379	380	385	386	386	389	389	390
	392	398	400	401	401	421	430	430	431	431
	435	438	438	438	439	448	448	448	449	453
	453	453	454	459	461	462	469	471	471	472
	480	481	481	486	491	491	494	496	496	499
	499	499	500	501	501	504	504	506	508	508
	509	509	513	514	514	515	519	522	527	529
	530	530	572	580	580	580	604	607	610	611
	719	722	724	724	725	725	796	859	866	866
	874	876	876	883	889	889	897	903	903	904
	904	908	909	909	915	915	919	920	920	923
	923	926	936	936	938	1195	1326	1328	1328	1329
	1329	1412	1412	1413	1413					
IncidentAlt	117	336	352	370	376	385	419	480	486	494
	501	501	506	513	519	522	570	585	589	598
	626	656	794	805	862	926	936	943	1152	1154
	1155	1162	1176	1176	1179	1179	1182	1402		
InputWinds	794	997	1184							
J	95	572	580	580	580	737	770	770	1412	1412
	1413	1413								
L	421	474	527							
LastDateTime	120	174	180	184	184	191	195	195	202	245
	1222	1223	1225	1402						
LastDegree	1196	1204	1205	1207	1207	1208	1213	1214	1215	1215
	1216									
LastLatitudeKnown	118	260	261	261	264	264	1204	1205	1403	
LastLongitudeKnown	119	277	278	278	281	281	1213	1214	1403	
LastMinute	1197	1205	1209	1209	1210	1214	1217	1217	1218	
LastMonth	97	227	228	228	231	231	1223	1225		
LastYear	98	236	1223	1225						
LatNS	90	252	254	254	254	254	1211	1403		
lines	147	153								
LongEW	91	269	271	271	271	271	1219	1404		
LowAltitude	341	348	368	372	385	424	431	448	459	471
	504	508	513							

MaxDistanceGlide	121	1013	1018	1028	1029	1046	1265	1404		
MaxRadiusGlide	122	1019	1049	1050	1266	1269	1404			
MethodDrift	92	584	589	616	623	1053	1065	1143	1287	1405
	1460	1460	1460							
N	96	355	380	390	392	398	461	469	474	519
	580	604	722	860	867	867	1326	1404		
No	137	754	1411	1454						
OnePassCompleted	84	593	658	694	710	1405				
P	797	801	928	928	929	936				
ParaDriftSpeed	123	627	630	1299	1405					
radians	80	298	607	671	677	697	700	1004	1008	1027
	1034									
RateParaDescent	124	652	656	1301						
ResultMagnitude	125	319	620	627	682	705	1013	1041	1406	
Temp	126	165	184	185	195	196	1078	1079	1096	1098
	1117	1176	1180							
Temp1	339	347	372	373	422	430	462	470	491	508
	509									
Temp2	340	348	385	386	423	431	481	486	491	513
	514	798	889	890						
TempAlt	574	619	620	626	627					
TempAngle	293	298	301	304	305	306	308	309	310	
TempDate	166	174	175	175	205	205				
TempHour	167	185	186	186	205	206				
TempMinutes	168	196	197	197	206	206				
TerrainHeight	127	584	862	1082	1084	1154	1182	1230	1232	1290
	1294	1406	1440	1442						
TimeParaDescent	128	656	1319	1406						
TotalBrg	129	681	686	704	1310	1407				

TotalDistance	130	682	687	705	1312	1407				
TotalRadius	131	690	706	1313	1316	1407				
TrajectBrg	132	700	1040	1045	1277	1408				
TrajectDistance	133	701	702	1041	1046	1279	1408			
TrajectRadius	134	706	1049	1050	1280	1283	1408			
TypeVehicle	99	557	559	559	561	562	563			
ValidAnswers	139	752	781	1139	1411	1428	1451			
VerifyDTG	163	246								
VerifyWinds	734	942								
Warranty	1339	1415								
WindChart	717	744	774							
WindCheck	736	741	750	752	754	779	781			
WindDisplacement	135	998	1005	1006	1018	1261	1409			
WindDrift	570	943	1054							
WindErrcr	738	741	759	761	761	763	767	768	769	770
Winds	82	142								
WindsAloft	142	580	724	725	725	767	768	769	770	842
	843	843	848	848	852	853	853	856	856	874
	876	876	883	889	889	897	903	903	904	904
	908	909	909	915	915	919	920	920	923	923
	926	936	1328	1329	1329	1412				
writelns	147	771	784	802	833	941	1416	1431		
WriteToDisk	1193	1461								
Xcomponent	291	298	301	302	303	307	319	319	321	575
	603	610	610	614	672	678	678	680	698	701
	701	703	956	960	1005	1009	1009	1011	1028	1037
	1037	1039								
Ycomponent	291	298	301	302	303	307	319	319	322	576
	603	611	611	614	673	679	679	680	699	702
	702	703	957	960	1006	1010	1010	1011	1029	1038
	1038	1039								

Yes	138 1141 1411 1429 1453

Operator	Program Line Number
arctan	298
assign	1201
boolean	84
char	92 139 736 1341
close	1332
cos	611 673 679 699 702 1006 1010 1029 1038
false	593 658 1405
input	78
integer	99 147 149 338 421 572 719 737 738 797 1195
output	78
read	770
readln	180 191 202 227 236 245 252 260 269 277 557 640 652 663 750 759 779 832 842 852 874 908 919 968 979 988 1075 1093 1137 1152 1389 1426 1440 1449
real	82 135 168 291 293 336 340 343 419 423 426 570 576 794 798 957 1197
rewrite	1202
round	1205 1214
sin	610 672 678 698 701 1005 1009 1028 1037
sqrt	319
text	1198
true	694 710
trunc	174 184 185 195 196 889 1078 1096 1176 1204 1213 1222

write	179	190	201	226	235	244	251	259	268	276
	394	400	401	529	530	556	629	639	651	662
	724	725	749	758	765	767	768	769	778	826
	831	836	841	851	873	878	892	907	911	918
	967	978	987	1068	1070	1072	1074	1092	1100	1106
	1112	1136	1146	1148	1151	1157	1206	1207	1208	1209
	1210	1211	1211	1215	1216	1217	1218	1229	1231	1233
	1235	1237	1241	1246	1251	1253	1256	1258	1260	1262
	1264	1268	1273	1276	1278	1282	1289	1293	1296	1298
	1300	1302	1306	1309	1311	1315	1318	1328	1329	1345
	1347	1349	1351	1353	1355	1357	1359	1361	1363	1365
	1367	1369	1371	1373	1375	1379	1381	1383	1385	1388
	1425	1439	1448							
writeln	155	177	178	188	189	199	200	217	219	220
	221	222	223	224	225	229	230	233	234	237
	239	240	241	242	243	247	250	253	257	258
	262	263	267	270	274	275	279	280	282	321
	322	323	395	396	397	402	404	524	525	526
	531	533	549	550	551	552	553	554	555	558
	628	630	631	632	633	634	635	636	637	638
	641	643	644	645	646	647	648	649	650	653
	661	665	666	726	747	748	751	757	760	766
	776	780	809	810	811	812	813	814	815	816
	817	818	819	820	821	822	823	824	825	827
	828	829	830	837	838	845	846	854	855	879
	880	885	886	893	894	899	900	912	913	921
	922	930	931	963	964	965	966	970	971	977
	981	982	986	990	991	1069	1071	1073	1080	1081
	1087	1088	1089	1090	1091	1101	1107	1113	1116	1134
	1135	1138	1147	1149	1150	1158	1159	1164	1165	1166
	1167	1168	1169	1170	1171	1172	1177	1219	1219	1223
	1225	1230	1232	1234	1236	1238	1242	1247	1252	1254
	1257	1259	1261	1263	1265	1269	1274	1277	1279	1283
	1290	1294	1297	1299	1301	1303	1307	1310	1312	1316
	1319	1323	1324	1330	1346	1348	1350	1352	1354	1356
	1358	1360	1362	1364	1366	1368	1370	1372	1374	1376
	1377	1378	1380	1382	1384	1386	1387	1417	1418	1419
	1420	1421	1422	1423	1427	1435	1436	1437	1438	1441
	1445	1446	1447	1450	1462	1463	1464	1465	1466	

117 Variables & Operators Used   1588 Occurrences

```
(*****************************************************************)
(*                                                               *)
(*          SEARCH PLANNING SOFTWARE (PROGRAM #2 OF 3)           *)
(*                                                               *)
(* TITLE:         SURFDRIF.COM (Surface Drift Algorithm)         *)
(* VERSION:       1.1  for CP/M Operating System                 *)
(* DATE WRITTEN:  September  1984                                *)
(* ------------------------------------------------------------- *)
(* DESCRIPTION:                                                  *)
(*   - User asked for last known position (coordinates) and      *)
(*     date-time-group of search object                          *)
(*   - User asked for desired Datum position (coordinates)       *)
(*     and date-time-group of search object                      *)
(*   - User must verify search object affected by oceanic        *)
(*     drift forces (object more than 20 miles off-shore and     *)
(*     in water deeper than 100 feet)                            *)
(*   - User asked to input reported surface winds in the         *)
(*     search object's vicinity for at least the last 48         *)
(*     hours                                                     *)
(*   - User must then input the wind latitude coefficient        *)
(*     directions and magnitudes for the object's latitude as    *)
(*     found in the National Search and Rescue (SAR) Manual      *)
(*   - Wind latitude coefficient vectors are printed back to     *)
(*     the user for verification/correction                      *)
(*   - User asked which of the three types of drift              *)
(*     uncertainty found in the National SAR Manual applies,     *)
(*     then the appropriate information is requested             *)
(*   - User is asked for the sea current vector and published    *)
(*     reference source it was found in                          *)
(*   - User is asked for the total observed water current, if    *)
(*     applicable                                                *)
(*   - User is asked for the tidal current, if applicable        *)
(*   - Program calculates applicable, resultant drift vector     *)
(*     direction(s) and magnitude(s) and creates an "audit       *)
(*     trail"/record file of program input, significant          *)
(*     calculations, and output, named "SEADATA"                 *)
(* ------------------------------------------------------------- *)
(* LICENSE:  COPYRIGHT 1984     D. RICK DOUGLAS                  *)
(*                                                               *)
(* The author makes no express or implied warranty of any        *)
(* kind with regard to this program material, including, but     *)
(* not limited to, the implied warranty of fitness for a         *)
(* particular purpose.  The author shall not be liable for       *)
(* incidental or consequential damages in connection with or     *)
(* arising out of furnishing, use, or performance of this        *)
(* program.  The reader MUST HAVE a solid understanding of       *)
(* search and rescue methodology before using this software      *)
(* in making decisions where human life is at risk.  In fact,    *)
(* since no amount of testing can uncover 100% of program        *)
(* errors, this program is recommended for training use only.    *)
(* Prior attendance at the United States Coast Guard's           *)
```

```
(*  National SAR School is highly-encouraged.                  *)
(*                                                             *)
(*        THIS  SOFTWARE  MAY  BE  FREELY-DISTRIBUTED          *)
(*        PROVIDED  NO  FEE  IS  CHARGED  AND                  *)
(*        THIS  COPYRIGHT  NOTICE  IS  RETAINED.               *)
(* ----------------------------------------------------------- *)
(*  LANGUAGE: PASCAL                                           *)
(*  USED    : Borland International, TURBO.PAS, Version 2.0    *)
(* ----------------------------------------------------------- *)
(*  MODULES CALLED (Sequentially listed); (OPT) = "Optional":  *)
(*                                                             *)
(*  SurfacePosition                                            *)
(*  VerifyDTG (Last Known Position)                            *)
(*  VerifyDTG (Datum Position)                                 *)
(*  WindPeriods                                                *)
(*  PericdTimes                                                *)
(*  DaysInMonth (OPT)                                          *)
(*  DaysInMonth (OPT)                                          *)
(*  DaysInMonth (OPT)                                          *)
(*  InputSeaWinds                                              *)
(*  WindLatCoeffs                                              *)
(*  VerifyWinds                                                *)
(*  WindChart                                                  *)
(*  WindChart (OPT)                                            *)
(*  WindCurrent                                                *)
(*  AddVectors                                                 *)
(*  AddVectors (OPT)                                           *)
(*  AvgSurfaceWind                                             *)
(*  AddVectors                                                 *)
(*  LeewayDrift                                                *)
(*  DriftDirUncertain (OPT)                                    *)
(*  SeaCurrent                                                 *)
(*  Datum                                                      *)
(*  WriteToDisk                                                *)
(*  RecordCurrents                                             *)
(*                                                             *)
(*************************************************************)



program SurfaceDrift(input,output);

const max = 4;                {Maximum number of wind current periods allowed}
      radians = 57.2957795;   {Standard radian conversion factor}

var continue,                 {Determines if this program should be used}
    LatNS,                    {Indicates whether latitude is North or South}
    LongEW       : char;      {Indicates whether longitude is East or West}

    DatumMonth,               {Month object will be at desired Datum position}
    DatumYear,                {Year object will be at desired Datum position}
    Days,                     {Number of days in specified month}
```

```
    DegreesLat,             {Lat known position latitude in degrees}
    I,J,K,                  {Used as counters}
    N,                      {Tracks number of 48-hr wind current periods used}
    HoursElapsed,           {Datum minus last known position time in hours}
    LastDay,                {Day object at last known position}
    LastHour,               {Hour object at last known position}
    LastMonth,              {Month object at last known position}
    LastYear,               {Year object at last known position}
    LeewayMethod,           {Indicates type of leeway drift uncertainty chosen}
    MaxDivergence,          {Maximum divergence from downwind in degrees}
    MaxHoursDrift,          {Maximum number of hours search object drifted}
    MinHoursDrift,          {Minimum number of hours search object drifted}
    SourceSeaCurr :integer; {Source used to determine sea current vector}

    AvgWindFrom,            {Average surface wind direction from (in degrees)}
    AvgWindTo,              {AvgWindFrom plus-or-minus 180}
    DatumDateTime,          {Day/Time object will be at desired Datum position}
    DirSeaCurrent,          {Sea current vector direction}
    DirSurfWind,            {Average surface wind direction}
    DirTidalCurrent,        {User input Tidal Current direction}
    DirTotalCurrent,        {Total water current vector direction}
    DmaxDir,                {Maximum combined surface drift direction}
    DminDir,                {Minimum combined surface drift direction}
    DmaxDistance,           {Maximum combined surface drift distance}
    DminDistance,           {Minimum combined surface drift distance}
    LastDateTime,           {Time object was at last known position}
    LastLatitudeKnown,      {Last known latitude of aerospace object}
    LastLongitudeKnown,     {Last known longitude of aerospace object}
    LeewayBrg,              {Average surface wind bearing}
    MaxDirWindCurrent,      {Total ocean surface current bearing due to wind}
    MinDirWindCurrent,      {Total ocean surface current bearing due to wind}
    MaxLeeBrg,              {Average surface wind bearing minus MaxDivergence}
    MinLeeBrg,              {Average surface wind bearing plus MaxDivergence}
    MaxLeeDistance,         {Maximum leeway drift distance of search object}
    MinLeeDistance,         {Minimum leeway drift distance of search object}
    MaxLeeSpeed,            {Maximum leeway drift rate of search object}
    MinLeeSpeed,            {Minimum leeway drift rate of search object}
    MaxSeaCurrDist,         {Maximum sea current drift distance}
    MinSeaCurrDist,         {Minimum sea current drift distance}
    MaxTideDistance,        {User input Tidal Current maximum distance}
    MinTideDistance,        {User input Tidal Current minimum distance}
    MaxTotCurrDist,         {Total water current vector maximum distance}
    MinTotCurrDist,         {Total water current vector minimum distance}
    MaxWindCurrDist,        {Total ocean surface current distance due to wind}
    MinWindCurrDist,        {Total ocean surface current distance due to wind}
    ResultMagnitude,        {Resultant distance computed from vector addition}
    SeaCurrSpeed,           {Sea current vector velocity}
    SurfWndMagnitude,       {Average surface wind speed times affected hours}
    SurfWndSpeed,           {Average surface wind in nautical miles per hour}
    TotCurrSpeed    : real; {Total water current vector velocity}

    No,                         {'N,n' = Legal "No"  characters}
```

```
    Yes,                          {'Y,y' = Legal "Yes" characters}
    ValidAnswers : set of char; {'Y,y,N,n' = Legal "Yes" and "No" characters}

    SeaDat          : text; {External record file of program input/output data}

    {Indicates the 1 to 6 hours of wind effect in each 48-hour period of   }
    {ocean surface current.                                                }

    HoursWindEffect  : array[1..max] of integer;


    {Indicates the velocity contributions of each 6-hour wind block times  }
    {the number of hours of wind effect in that block to calculate average }
    {surface wind.                                                         }

    VelocityComponent : array[1..max] of real;


    {Indicates the wind current latitude coefficient vector directions     }
    {(position 1) and magnitudes (position 2) from the National SAR Manual.}

    WindCoeffs       : array[1..8,1..3] of real;


    {Wind vectors affecting ocean surface current over time:  First position}
    {indicates the number of 48-hour periods;   Second position indicates   }
    {where the reported, 6-hour wind vector fits amongst the 8 times given  }
    {in each 48-hour period; and,   Third position indicates the days (1),  }
    {hours (2), directions (3), and velocities (4) of each 6-hour wind      }
    {vector, as well as the sum of these vectors and the Wind Current       }
    {Latitude Coefficient vectors directions (5) and magnitudes (6).        }

    WindsOverTime    : array[1..max,1..8,1..6] of real;

    {The total contribution (sum) of WindsOverTime and WindCoeffs vector     }
    {directions and velocities for the eight intervals per period.           }

    PeriodVectors : array[1..max,1..3] of real;

{************************************************************************}
{Writes out a specified number of blank lines; can be used to clear screen.  }

procedure writelns (lines : integer);

var count : integer;

begin
   count := 0;
   while lines > count do
   begin
      writeln;
      count := count + 1
```

```
    end
end;

{******************************************************************************}
{Verifies legitimate date/time/group data input.}

procedure VerifyDTG (var DateTime : real);

var Temp,               {Used as a temporary computation variable}
    TempDate,           {Used as temporary date variable}
    TempHour,           {Used as temporary hour variable}
    TempMinutes : real; {Used as temporary minutes variable}

begin  {VerifyDTG}
   repeat {until correct date/time format input}

      {Verify the day is between 1 and 31}
      TempDate := trunc( DateTime/10000 );
      if (TempDate < 1) or (TempDate > 31) then
      begin  {TempDate not between 1 and 31}
         writeln;
         writeln('You have incorrectly entered the date.  Try again!');
         write('Re-enter Z DTG=> ');
         readln(DateTime)
      end;  {TempDate not between 1 and 31}

      {Verify the hour is between 0000 and 2400}
      Temp := (DateTime/10000 - (trunc(DateTime/10000)) ) * 100;
      TempHour := trunc(Temp);
      if (TempHour < 0) or (TempHour > 23) then
      begin  {TempHour not between 0 and 23}
         writeln;
         writeln('You have incorrectly entered the hour.  Try again!');
         write('Re-enter Z DTG=> ');
         readln(DateTime)
      end;  {TempHour not between 0 and 23}

      {Verify there are no minutes, e.g. minutes = 0}
      Temp := (DateTime/100 - (trunc(DateTime/100)) ) * 100;
      TempMinutes := trunc(Temp);
      if TempMinutes <> 0 then
      begin  {TempMinutes not 0}
         writeln;
         writeln('Please enter the date/time to the nearest hour.  Try again!');
         write('Re-enter Z DTG=> ');
         readln(DateTime)
      end    {TempMinutes not 0}

   until (TempDate >= 1) and (TempDate <= 31) and (TempHour >= 0) and
         (TempHour <= 23) and (TempMinutes = 0)

end;  {VerifyDTG}
```

```
{*************************************************************************}
{Asks user for the last known and the desired Datum times and positions of a }
{search object on the ocean's surface.                                        }

procedure SurfacePosition;

var Temp : real;     {Used as a temporary computation variable}

begin  {SurfacePosition}

   writeln;
   repeat  {until valid response}
      writeln('Please enter the number for the month the search object was');
      writeln('at the last known position:');
      writeln;
      writeln(' 1 = Jan        4 = Apr        7 = Jul        10 = Oct');
      writeln(' 2 = Feb        5 = May        8 = Aug        11 = Nov');
      writeln(' 3 = Mar        6 = Jun        9 = Sep        12 = Dec');
      writeln;
      write('LAST KNOWN POSITION MONTH=> ');
      readln(LastMonth);
      if (LastMonth < 1) or (LastMonth > 12) then
         writeln('Incorrect number entered.  Please try again!');
      writeln
   until (LastMonth >= 1) and (LastMonth <= 12);

   writeln('Please enter the year the search object was at the last known');
   writeln('position  (e.g., 1985, 1986, etc.)');
   write('LAST KNOWN POSITION YEAR=> ');
   readln(LastYear);
   writeln;

   CLRSCR;
   writeln('Please enter the day-hour-minute (TO THE NEAREST HOUR) the');
   writeln('search object was at the last known position.  Enter it in');
   writeln('Z DTG (Zulu Date-Time-Group) format.  For example: 9:37 PM,');
   writeln('August 4, should appear as 042200 Greenwich-Mean Time (Z=0).');
   writelns(2);
   writeln('If the number of hours of search object drift is uncertain,');
   writeln('you must run this program twice (as mentioned earlier).  On');
   writeln('the first run enter here the time the SHORTER drift period');
   writeln('started (LATER than the last known position time).  On the');
   writeln('second run, or if two runs are not applicable, enter here the');
   writeln('the LKP time as requested above:');
   write('LAST KNOWN POSITION TIME (Z DTG)=> ');
   readln(LastDateTime);
   VerifyDTG(LastDateTime);

   Temp := LastDateTime/10000;
   LastDay := trunc(Temp);
   LastHour := (round ((LastDay - Temp) * -100) ) * 100;  {nearest hundred}
```

```
CLRSCR;
repeat {until valid latitude}
   writeln('Was search object''s last known latitude north or south?');
   write('(Enter N or S)      Answer=) ');
   readln(LatNS);
   writeln
until (LatNS = 'N') or (LatNS = 'n') or (LatNS = 'S') or (LatNS = 's');
if (LatNS = 'N') or (LatNS = 'n') then LatNS := 'N' else LatNS := 'S';

repeat {until valid latitude}
   writeln('Please enter the search object''s last known latitude ');
   writeln('(For example: 25-Degrees 45-Minutes 13-Seconds = 25.4513)');
   write('LATITUDE =) ');
   readln(LastLatitudeKnown);
   if (LastLatitudeKnown < 0) or (LastLatitudeKnown > 90) then
      writeln('Input latitude must be between 0-90.  Try again!');
   writeln
until (LastLatitudeKnown >= 0) and (LastLatitudeKnown <= 90);
DegreesLat := trunc(LastLatitudeKnown);

repeat {until valid longitude}
   writeln('Was search object''s last known longitude east or west?');
   write('(Enter E or W)      Answer=) ');
   readln(LongEW);
   writeln
until (LongEW = 'E') or (LongEW = 'e') or (LongEW = 'W') or (LongEW = 'w');
if (LongEW = 'E') or (LongEW = 'e') then LongEW := 'E' else LongEW := 'W';

repeat {until valid longitude}
   writeln('Please enter the search object''s last known longitude ');
   writeln('(For example: 160-Degrees 45-Minutes 13-Seconds = 160.4513)');
   write('LONGITUDE =) ');
   readln(LastLongitudeKnown);
   if (LastLongitudeKnown < 0) or (LastLongitudeKnown > 180) then
      writeln('Input longitude must be between 0-180.  Try again!');
   writeln
until (LastLongitudeKnown >= 0) and (LastLongitudeKnown <= 180);

CLRSCR;
writeln('Now you will be asked to enter the year, month, and date/time for');
writeln('the desired Datum position, which MUST be later than the time you');
writeln('just entered for the last known position!');
writeln;

repeat  {until valid response}
   writeln('Please enter the desired Datum month:');
   writeln;
   writeln(' 1 = Jan         4 = Apr         7 = Jul         10 = Oct');
   writeln(' 2 = Feb         5 = May         8 = Aug         11 = Nov');
   writeln(' 3 = Mar         6 = Jun         9 = Sep         12 = Dec');
   writeln;
```

```
      write('DATUM MONTH=) ');
      readln(DatumMonth);
      if (DatumMonth ( 1) or (DatumMonth ) 12) then
         writeln('Incorrect number entered.  Please try again!');
      writeln
   until (DatumMonth )= 1) and (DatumMonth (= 12);

   writeln('Please enter the desired Datum year (e.g., 1985, 1986, etc.)');
   write('DATUM YEAR=) ');
   readln(DatumYear);
   writeln;

   writeln;
   write('Please enter the desired Datum day-hour-minute ');
   writeln('(TO THE NEAREST HOUR).');
   writeln('Enter it in Z DTG (Zulu Date-Time-Group) format.  For example:');
   write('9:37 PM, August 4, should appear as 042200 ');
   writeln('Greenwich-Mean Time (Z=0).');
   write('DATUM TIME (Z DTG)=) ');
   readln(DatumDateTime);
   VerifyDTG(DatumDateTime);
   writeln;

   Temp := DatumDateTime/10000

end;  {SurfacePosition}

{************************************************************************}
{Given the last known surface position time, and the hours elapsed from then }
{until the desired Datum time, this procedure determines the number of      }
{48-hour wind-current-effect periods, and the number of hours (1 to 6) in    }
{each period, required to calculate the average ocean surface current caused }
{by the wind.                                                               }

procedure WindPeriods;

var HoursRemaining : integer; {Elapsed hours minus those used in calculations}

begin  {WindPeriods}

   {Find number of hours in first 48-hour period}
   if (LastHour )= 0000) and (LastHour ( 0300) then
      HoursWindEffect[1] := 3 - round(LastHour/100);
   if (LastHour )= 0300) and (LastHour ( 0900) then
      HoursWindEffect[1] := 9 - round(LastHour/100);
   if (LastHour )= 0900) and (LastHour ( 1500) then
      HoursWindEffect[1] := 15 - round(LastHour/100);
   if (LastHour )= 1500) and (LastHour ( 2100) then
      HoursWindEffect[1] := 21 - round(LastHour/100);
   if LastHour )= 2100 then
      HoursWindEffect[1] := (24 - round(LastHour/100) ) + 3;

```

```
   N := 1;  {initialize number of 48-hour periods}
   HoursRemaining := HoursElapsed - HoursWindEffect[1];
   if HoursRemaining > 0 then
   begin  {More 48-hour periods are required}

      repeat  {until HoursRemaining = 0}
         N := N + 1;
         if HoursRemaining >= 6 then
         begin  {More than six hours remain between current and Datum hour}
            HoursWindEffect[N] := 6;
            HoursRemaining := HoursRemaining - 6
         end   {More than six hours remain between current and Datum hour}
         else
         begin  {One to five hours remain between current and Datum hour}
            HoursWindEffect[N] := HoursRemaining;
            HoursRemaining := 0
         end   {One to five hours remain between current and Datum hour}
      until HoursRemaining = 0
   end  {More 48-hour periods are required}
end;  {WindPeriods}

{************************************************************************}
{Given the month, determines the number of days in that month, including a   }
{check to see if that month is a leap-year-February with 29 days.            }

procedure DaysInMonth (TempMonth : integer);

var LeapYearCheck,          {If zero, specified year is a leap year}
    TempYearDiv    : real;  {Result of specified year divided by 4}

begin  {DaysInMonth}
   if TempMonth = 2  {February}  then
   begin  {Leap year check}
      TempYearDiv := LastYear/4;
      LeapYearCheck := TempYearDiv - trunc(TempYearDiv);
      if LeapYearCheck = 0 then TempMonth := 13  {Leap year February}
   end;  {Leap year check}

   case TempMonth of
      1  : Days := 31;
      2  : Days := 28;
      3  : Days := 31;
      4  : Days := 30;
      5  : Days := 31;
      6  : Days := 30;
      7  : Days := 31;
      8  : Days := 31;
      9  : Days := 30;
      10 : Days := 31;
      11 : Days := 30;
      12 : Days := 31;
      13 : Days := 29  {<- Leap year February}
```

```
   end  {case of number of days in specified month}

end;  {DaysInMonth}

{*************************************************************************}
{Determines the day and hour for which wind directions and velocities are  }
{required in the eight intervals of each period to calculate each period's }
{ocean surface wind current component vector.                              }

procedure PeriodTimes;

var I,J,                      {Used as counters}
    MonthBefore : integer;  {The month before the last known position month}

begin  {PeriodTimes}

   {Find day and time for period #1, interval #1}
   WindsOverTime[1,1,1] := LastDay;  {initialize at last known position day}
   if LastHour = 0000  {midnight}  then WindsOverTime[1,1,2] := 0000;
   if (LastHour >= 0100) and (LastHour <= 0600) then
                                              WindsOverTime[1,1,2] := 0600;
   if (LastHour >= 0700) and (LastHour <= 1200) then
                                              WindsOverTime[1,1,2] := 1200;
   if (LastHour >= 1300) and (LastHour <= 1800) then
                                              WindsOverTime[1,1,2] := 1800;

   if (LastHour >= 1900) and (LastHour < 2400) then
   {Find next day and start with time = 0000}
   begin  {LastHour = 7-11 P.M.}
      DaysInMonth(LastMonth);
      if LastDay < Days then
      begin  {Last known position-day is NOT the last day of the month}
         WindsOverTime[1,1,1] := LastDay + 1;
         WindsOverTime[1,1,2] := 0000
      end   {Last known position-day is NOT the last day of the month}
      else
      begin  {Last known position-day is the last day of the month}
         WindsOverTime[1,1,1] := 1;
         WindsOverTime[1,1,2] := 0000
      end   {Last known position-day is the last day of the month}
   end;  {LastHour = 7-11 P.M.}

   for J := 2 to 8 do
   begin  {Find days/times of remaining 7, 6-hr intervals in the 48-hr-period}

      if WindsOverTime[1,J-1,2] > 0000  {not equal to midnight} then
      begin  {this period's previous time = 6 A.M. or P.M. or 12 P.M.}

         {Next interval's date equals the same as the previous interval's date}
         WindsOverTime[1,J,1] := WindsOverTime[1,J-1,1];

         {Next interval's hour equals last interval's hour minus six hours}
```

```
          WindsOverTime[1,J,2] := WindsOverTime[1,J-1,2] - 0600

        end   {this period's previous time = 6 A.M. or P.M. or 12 P.M.}
        else
        begin  {this period's previous time = midnight}
          if WindsOverTime[1,J-1,1] > 1 then {Day of month is 2nd through last}
          begin  {Set period's interval time to 1800 of previous day}
              WindsOverTime[1,J,1] := WindsOverTime[1,J-1,1] - 1;
              WindsOverTime[1,J,2] := 1800
          end    {Set period's interval time to 1800 of previous day}
          else  {First day of month}
          begin  {Determine previous month's last day and set at 6 P.M.}
            if LastMonth > 1 then MonthBefore := LastMonth - 1
            else MonthBefore := 12;
            DaysInMonth(MonthBefore);
              WindsOverTime[1,J,1] := Days;
              WindsOverTime[1,J,2] := 1800
          end    {Determine previous month's last day and set at 6 P.M.}
        end    {this period's previous time = midnight}
      end;   {Find days/times of remaining 7, 6-hr intervals in the 48-hr-period}

      if N > 1 then
      begin  {More than one period}
        I := 1;  {initialize}

        repeat  {until I = N}
          I := I + 1;

          {Find days and times of intervals in each period}
          WindsOverTime[I,1,1] := WindsOverTime[I-1,1,1];  {initialize}
          if WindsOverTime[I-1,1,2] = 0000 then WindsOverTime[I,1,2] := 0600;
          if WindsOverTime[I-1,1,2] = 0600 then WindsOverTime[I,1,2] := 1200;
          if WindsOverTime[I-1,1,2] = 1200 then WindsOverTime[I,1,2] := 1800;
          if WindsOverTime[I-1,1,2] = 1800 then
          begin  {Find next day and start with 0000}
            DaysInMonth(LastMonth);

            {if not last day of month, add a day}
            if WindsOverTime[I-1,1,1] < Days then
              WindsOverTime[I,1,1] := WindsOverTime[I-1,1,1] + 1

            {if last day of month, set to first day (of next month)}
            else WindsOverTime[I,1,1] := 1;

            WindsOverTime[I,1,2] := 0000
          end;  {Find next day and start with 0000}

          for J := 2 to 8 do
          begin  {fill in days and times of each period's other seven intervals}
            WindsOverTime[I,J,1] := WindsOverTime[I-1,J-1,1];
            WindsOverTime[I,J,2] := WindsOverTime[I-1,J-1,2]
          end    {fill in days and times of each period's other seven intervals}
```

```
      until I = N
    end    {More than one period}
end;   {PeriodTimes}

{*****************************************************************}
{Queries user for ocean surface wind directions and velocities.            }

procedure InputSeaWinds;

var J : integer;  {Used as a counter}

begin  {InputSeaWinds}
   CLRSCR;

   for J := 8 downto 1 do
      begin  {Enter individual ocean surface wind vectors}
         write('Please enter the ocean surface wind direction and ');
         if WindsOverTime[1,J,1] < 10 then
            write('velocity at 0',WindsOverTime[1,J,1]:1:0)
         else write('velocity at ',WindsOverTime[1,J,1]:2:0);
         if WindsOverTime[1,J,2] = 0000 then
            writeln('000',WindsOverTime[1,J,2]:1:0,'Z:')
         else if WindsOverTime[1,J,2] = 0600 then
            writeln('0',WindsOverTime[1,J,2]:3:0,'Z:')
         else writeln(WindsOverTime[1,J,2]:4:0,'Z:');

         repeat  {until degrees = 0 to 360}
            write('WIND DIRECTION = ');
            readln(WindsOverTime[1,J,3])
         until (WindsOverTime[1,J,3] >= 0) and (WindsOverTime[1,J,3] <= 360);

         repeat  {until velocity = 0 to 99}
            write('WIND VELOCITY = ');
            readln(WindsOverTime[1,J,4]);
            writeln
         until (WindsOverTime[1,J,4] >= 0) and (WindsOverTime[1,J,4] <= 99)
      end;  {Enter individual ocean surface wind vectors}

      if N > 1 then  {Checks for more than one 48-hour wind period}
      begin  {more than one period}
         I := 1;
         {Input wind vectors from last interval in each period}
         repeat  {until I = N}
            I := I + 1;
            write('Please enter the ocean surface wind direction and ');
            write('velocity at ',WindsOverTime[I,1,1]:2:0);
            if WindsOverTime[I,1,2] = 0000 then
               writeln('000',WindsOverTime[I,1,2]:1:0,'Z:')
            else if WindsOverTime[I,1,2] = 0600 then
               writeln('0',WindsOverTime[I,1,2]:3:0,'Z:')
            else writeln(WindsOverTime[I,1,2]:4:0,'Z:');

```

```
            repeat  {until degrees = 0 to 360}
              write('WIND DIRECTION = ');
              readln(WindsOverTime[I,1,3])
            until (WindsOverTime[I,1,3] )= 0) and
                  (WindsOverTime[I,1,3] (= 360);

            repeat  {until velocity = 0 to 99}
              write('WIND VELOCITY = ');
              readln(WindsOverTime[I,1,4]);
              writeln
            until (WindsOverTime[I,1,4] )= 0) and
                  (WindsOverTime[I,1,4] (= 99);

            for J := 2 to 8 do
            begin  {fill in wind vectors of each period's last 7 intervals}
              WindsOverTime[I,J,3] := WindsOverTime[I-1,J-1,3];
              WindsOverTime[I,J,4] := WindsOverTime[I-1,J-1,4]
            end    {fill in wind vectors of each period's last 7 intervals}
          until I = N
      end    {more than one period}
end;  {InputSeaWinds}

{*************************************************************************}
{Prints keyboard-input wind coefficient directions and velocities on the}
{video screen for user verification.}

procedure WindChart;

var I : integer;  {Used as a counter}

begin  {WindChart}
   write('Wind Current Latitude Coefficients for ',DegreesLat:2,'-degrees ');
   if LatNS = 'N' then writeln('North are:') else writeln('South are:');
   writeln;

   for I := 1 to 8 do
   begin  {for I = 1 to number of individual wind vectors}
     write('     ',I:1,'.     Period #',I:1,'   ',WindCoeffs[I,1]:3:0);
     writeln(' / ',WindCoeffs[I,2]:4:3)
   end;  {for I = 1 to number of individual wind vectors}

   writeln
end;  {WindChart}

{*************************************************************************}
{Allows user to change ocean surface wind directions and velocities entered  }
{in procedure InputSeaWinds.}

procedure VerifyWinds;

var WindCheck : char;     {Used to verify input wind data}
    J         : integer;  {Used as a counter}
```

```
    WindError : integer;  {Used to indicate which input wind data is in error}

begin  {VerifyWinds}
   WindCheck := 'Y';  WindError := -1;  {initialize}

{Verify input wind directions and velocities.}
   WindChart;  {Prints current wind data on video screen}

   repeat  {until WindCheck = valid response}
      writeln;
      writeln('Are these altitudes, wind directions, and wind');
      write('velocities all correct? (y/n) ');
      readln(WindCheck);
      writeln
   until (WindCheck in ValidAnswers);

   while (WindCheck in No) do
   begin  {while WindCheck = NO}
      repeat  {until WindError = 0 to 20}
         writeln('Which line is in error?');
         write('Enter number, or zero for none) =) ');
         readln(WindError);
         writeln
      until (WindError )= 0.0) and (WindError ( 21.0);

      if WindError ) 0.0 then
      begin  {if an input line is identified as being in error}
         write('Please enter altitude, wind direction ');
         writeln('and velocity to replace:');
         write(WindCoeffs[WindError,1]:4:0,' ');
         write(WindCoeffs[WindError,2]:4:0);
         write(' ',WindCoeffs[WindError,3]:4:0,' =) ');
         for J := 1 to 2 do read(WindCoeffs[WindError,J]);
         CLRSCR
      end;  {if an input line is identified as being in error}

      WindChart;  {Prints current wind data on video screen}

      writeln;
      repeat  {until valid WindCheck response}
         write('Are all other lines correct? (y/n) ');
         readln(WindCheck);
         writeln
      until (WindCheck in ValidAnswers)

   end;  {while WindCheck = NO}
   CLRSCR
end;  {VerifyWinds}

{*************************************************************************}
{When given X and Ycomponents this procedure calculates resultant vector    }
{bearing (AvgWindFrom) and magnitude (ResultMagnitude).                     }
```

```
procedure AddVectors (var Xcomponent, Ycomponent : real);

var TempAngle    : real;  {Temporary calculation variable}

begin  {AddVectors}

   {Find resultant angle (in degrees) uncorrected for compass position.}
   TempAngle := arctan(Xcomponent/Ycomponent) * radians;

   {Find bearing resulting from TempAngle corrected for compass position.}
   if (Xcomponent > 0) and (Ycomponent > 0) then AvgWindFrom := TempAngle;
   if ((Xcomponent > 0) and (Ycomponent < 0)) or
      ((Xcomponent < 0) and (Ycomponent < 0)) then
      if (TempAngle + 180) > 360 then
         AvgWindFrom := TempAngle - 180
      else AvgWindFrom := TempAngle + 180;
   if (Xcomponent < 0) and (Ycomponent > 0) then
      if (TempAngle + 360) > 360 then
         AvgWindFrom := TempAngle - 360
      else AvgWindFrom := TempAngle + 360;

   {Convert AvgWindFrom to AvgWindTo bearing.}
   if (AvgWindFrom + 180) > 360 then
      AvgWindTo := AvgWindFrom - 180
   else AvgWindTo := AvgWindFrom + 180;

   {Resultant bearing magnitude is the square root of the sum of}
   { the squared components.}
   ResultMagnitude := sqrt(Xcomponent * Xcomponent  +  Ycomponent * Ycomponent);

   writeln('Xcomponent = ',Xcomponent:5:3);
   writeln('Ycomponent = ',Ycomponent:5:3);
   writeln
end;  {AddVectors}

{**********************************************************************}
{Queries user for Wind Latitude Coefficient Vector directions and velocities }
{from National SAR Manual tables.                                            }

procedure WindLatCoeffs;

var I : integer;  {Used as a counter}

begin  {WindLatCoeffs}
   if DegreesLat >= 5 then
   begin  {Last known position's latitude >= 5 degrees}
      CLRSCR;
      write('Refer to the latitude coefficient-vectors table for the ');
      if LatNS = 'N' then writeln('Northern') else writeln('Southern');
      write('latitudes (approximately page ');
      if LatNS = 'N' then writeln('8-16c) in the National SAR Manual.')
```

```
        else writeln('8-16d) in the National SAR Manual.');
        write('Find the degrees-latitude column nearest to ',DegreesLat:2);
        write('-degrees-');
        if LatNS = 'N' then writeln('North.') else writeln('South.');
        write('You will now be asked to enter the directions and ');
        writeln('magnitudes for');
        writeln('each of the eight periods appearing under that column.');
        writeln;

        for I := 1 to 8 do
        begin  {Input applicable latitude coefficient vectors for the 8 periods}

           repeat  {until valid compass direction}
              write('PERIOD #',I:1,' DIRECTION = ');
              readln(WindCoeffs[I,1])
           until (WindCoeffs[I,1] >= 0.0) and (WindCoeffs[I,1] <= 360.0);

           repeat  {until valid compass direction}
              write('PERIOD #',I:1,' MAGNITUDE = ');
              readln(WindCoeffs[I,2])
           until (WindCoeffs[I,2] >= 0.003) and
                 (WindCoeffs[I,2] <= 0.03);

           writeln
        end;  {Input applicable latitude coefficient vectors for the 8 periods}
        CLRSCR;
        VerifyWinds
     end    {Last known position's latitude >= 5 degrees}

     else
     {If last known position's latitude is less than 5-degrees, then the eight}
     {Wind Current Latitude Coefficient directions remain at their initialized}
     {value of zero, and their magnitudes are set to one.  This is done to    }
     {ensure each periods overall wind current vector is unaffected until     }
     {later, when 5% of their velocity value will be determined as the total  }
     {wind current vector of all combined periods as directed in the National }
     {SAR Manual, approximately pages 8-16b, 8-16c, and 8-16d.                }

     for I := 1 to 8 do WindCoeffs[I,2] := 1

  end;  {WindLatCoeffs}

  {***********************************************************************}
  {Uses vector addition to calculate the resultant wind current vector     }
  {direction and magnitude for each period and for the sum of all period   }
  {vectors.                                                                }

  procedure WindCurrent;

  var I,J         : integer;  {Used as counters}
      BrgRads,                {Individual bearings converted to radians}
      Xcomponent,             {Horizontal component of individual vectors}
```

```
   Ycomponent : real;    {Vertical component of individual vectors}

begin  {WindCurrent}
  for I := 1 to N do
  begin  {repeat calculations for each period}
    Xcomponent := 0.0;  Ycomponent := 0.0;  {initialize}

    for J := 1 to 8 do
    begin  {find total contribution of WindsOverTime and WindCoeffs}
      WindsOverTime[I,J,5] := WindsOverTime[I,J,3] + WindCoeffs[J,1];
      WindsOverTime[I,J,6] := WindsOverTime[I,J,4] * WindCoeffs[J,2];
      BrgRads := WindsOverTime[I,J,5] / radians;
      Xcomponent := Xcomponent + sin(BrgRads) * WindsOverTime[I,J,6];
      Ycomponent := Ycomponent + cos(BrgRads) * WindsOverTime[I,J,6]
    end;  {find total contribution of WindsOverTime and WindCoeffs}

    writeln('Period #',I:1);
    AddVectors(Xcomponent,Ycomponent);
    PeriodVectors[I,1] := AvgWindFrom;
    PeriodVectors[I,2] := ResultMagnitude;
    PeriodVectors[I,3] := PeriodVectors[I,2] * HoursWindEffect[I]
  end;  {repeat calculations for each period}

  if N = 1 then  {Only one 48-hour wind period}
  begin  {Resultant wind current vector = resultant period #1 vector}
    MaxDirWindCurrent := PeriodVectors[1,1];
    MaxWindCurrDist := PeriodVectors[1,3]
  end   {Resultant wind current vector = resultant period #1 vector}

  else
  begin  {There are more than one 48-hour wind periods}
    Xcomponent := 0.0;  Ycomponent := 0.0;  {re-initialize}
    for I := 1 to N do
    begin  {Find each PeriodVector's Xcomponent and Ycomponent}
      BrgRads := PeriodVectors[I,1] / radians;
      Xcomponent := Xcomponent + sin(BrgRads) * PeriodVectors[I,3];
      Ycomponent := Ycomponent + cos(BrgRads) * PeriodVectors[I,3]
    end;  {Find each PeriodVector's Xcomponent and Ycomponent}

    writeln('Total Wind Current:');
    AddVectors(Xcomponent,Ycomponent);
    MaxDirWindCurrent := AvgWindFrom;
    MaxWindCurrDist := ResultMagnitude
  end;  {There are more than one 48-hour wind periods}

  {if last known position's latitude < 5-degrees, then only 5% of the total}
  {downwind drift distance applies as referenced above from the National   }
  {SAR Manual.                                                              }
  if DegreesLat < 5 then MaxWindCurrDist := 0.05 * MaxWindCurrDist;

  write('HIT RETURN (Once or twice, as necessary) TO CONTINUE');
  readln(continue);
```

```
    CLRSCR;

    write('Total Wind Current Direction is ',MaxDirWindCurrent:3:0,'-degrees ');
    writeln('for ',MaxWindCurrDist:4:2,' nautical miles.');
    writelns(2);
    writeln('If the number of hours of search object drift is uncertain, you');
    writeln('must run this program twice (as mentioned earlier).  On the first');
    writeln('run, RECORD this Total Wind Current vector direction and distance');
    writeln('for input during the second run.');
    writeln;
    writeln('If this is the second run, please now ENTER the previously-');
    writeln('recorded Wind Current vector over the shorter drift period');
    writeln('(If not applicable, enter a heading of 361):');
    repeat  {until valid compass heading or 361}
      write('WIND CURRENT DIRECTION (from Run #1) => ');
      readln(MinDirWindCurrent)
    until (MinDirWindCurrent >= 0) and (MinDirWindCurrent <= 361);

    if MinDirWindCurrent <> 361 then
      repeat  {until valid response}
        write('WIND CURRENT DISTANCE (from Run #1) => ');
        readln(MinWindCurrDist)
      until MinWindCurrDist >= 0

end;   {WindCurrent}

{**************************************************************************}
{Determines the average surface (leeway) wind blowing on the search object's }
{exposed area above the ocean's surface.                                    }

procedure AvgSurfaceWind;

var I          : integer;  {Used as counter}
    BrgRads,               {Individual bearings converted to radians}
    Xcomponent,            {Horizontal component of individual vectors}
    Ycomponent : real;     {Vertical component of individual vectors}

begin {AvgSurfaceWind}
   Xcomponent := 0.0;  Ycomponent := 0.0;  {re-initialize}

   for I := 1 to N do
   begin  {Find each 6-hour wind block's Xcomponent and Ycomponent}
      VelocityComponent[I] := WindsOverTime[I,1,4] * HoursWindEffect[I];
      BrgRads := WindsOverTime[I,1,3] / radians;
      Xcomponent := Xcomponent + sin(BrgRads) * VelocityComponent[I];
      Ycomponent := Ycomponent + cos(BrgRads) * VelocityComponent[I]
   end;  {Find each 6-hour wind block's Xcomponent and Ycomponent}

   CLRSCR;
   writeln('Average Surface Wind:');
   AddVectors(Xcomponent,Ycomponent);
   LeewayBrg := AvgWindTo;
```

```
   DirSurfWind := AvgWindFrom;
   SurfWndMagnitude := ResultMagnitude;
   SurfWndSpeed := SurfWndMagnitude / HoursElapsed;
   write('Average Surface Wind Direction ');
   write(DirSurfWind:3:0,'-degrees True at ');
   writeln(SurfWndSpeed:4:2,' knots.');
   writelns(2);
   writeln('If the number of hours of search object drift is uncertain, you');
   writeln('must run this program twice (as mentioned earlier).  On the first');
   writeln('run, RECORD this Average Surface Wind vector direction and');
   writeln('speed for use in calculating the minimum leeway speed during the');
   writeln('second program run.  Then, ABORT this program run, CLEAR your');
   writeln('microcomputer''s memory, and RERUN the entire program');
   writelns(6);
   write('HIT RETURN (Once or twice, as necessary) TO CONTINUE ');
   readln(continue)

end;  {AvgSurfaceWind}

{*********************************************************************}
{Used by procedure Leeway (if drift rate and time are known with certainty)  }
{to calculate minimum and maximum leeway drift direction.                    }

procedure DriftDirUncertain;

begin  {Directional Drift Uncertainty}

   writeln('Refer to the Leeway Speed Graph in the National SAR Manual');
   writeln('(approximately page 8-15).  Please enter the search object''s');
   writeln('maximum expected degrees-divergence from the downwind vector:');
   repeat  {until ) 0}
      write('MAX EXPECTED DIVERGENCE =) ');
      readln(MaxDivergence)
   until (MaxDivergence )= 0) and (MaxDivergence (= 60);

   if (LeewayBrg - MaxDivergence) ( 0 then    {adjusts for compass limits}
      MinLeeBrg := (LeewayBrg + 360.0) - MaxDivergence
   else MinLeeBrg := LeewayBrg - MaxDivergence;

   if (LeewayBrg + MaxDivergence) ) 360 then  {adjusts for compass limits}
      MaxLeeBrg := (LeewayBrg + MaxDivergence) - 360.0
   else MaxLeeBrg := LeewayBrg + MaxDivergence;

   writeln;
   writeln('Refer to the Leeway Speed Formulae in the National SAR Manual');
   writeln('(approximately page 8-13).  Please enter the leeway speed at');
   writeln('which the search object drifts:');
   repeat  {until ) 0}
      write('LEEWAY SPEED =) ');
      readln(MaxLeeSpeed)
   until MaxLeeSpeed ) 0;
```

```
   writeln;
   writeln('Please enter the number of hours the search object drifted:');
   repeat  {until > 0}
     write('HOURS OF DRIFT => ');
     readln(MaxHoursDrift)
   until MaxHoursDrift > 0;

   writeln;
   MinLeeDistance := MaxLeeSpeed * MaxHoursDrift;
   MaxLeeDistance := MinLeeDistance

end;  {Directional Drift Uncertainty}

{*************************************************************************}
{Calculates the total leeway drift direction and distance.                }

procedure LeewayDrift;

begin  {LeewayDrift}

   CLRSCR;
   write('This program will now calculate the LEEWAY vector, ');
   writeln('or, that drift caused');
   write('by average surface winds pushing on the exposed ');
   writeln('area of the search object.');
   writeln('There are three leeway vector calculation options:');
   writeln;
   write('(1)  DRIFT RATE UNCERTAINTY - Used when the search ');
   writeln('object is unknown,');
   write('     or, when it is not known whether the object ');
   writeln('has deployed an');
   write('     anti-drift device (e.g., sea drogue).  All ');
   writeln('drift is computed as');
   writeln('      a minimum and maximum distance downwind;');
   writeln;
   write('(2)  DRIFT TIME UNCERTAINTY - Used if the number ');
   writeln('of hours the search');
   write('     object has been drifting is not known for ');
   writeln('certain.  All drift');
   writeln('      is computed as a minimum and maximum distance downwind;');
   writeln;
   write('(3)  DIRECTIONAL DRIFT UNCERTAINTY - Used if above ');
   writeln('options do not');
   write('     apply.  However, drift is computed for a ');
   writeln('divergent bearing left');
   writeln('      and right of the downwind vector.');
   writeln;
   writeln('Refer to the National SAR Manual for additional information.');
   writeln;
   writeln;
   writeln;
   writeln;
```

```
   repeat  {until valid response}
      write('PLEASE SELECT OPTION #1, #2, or #3 => ');
      readln(LeewayMethod)
   until (LeewayMethod >= 1) and (LeewayMethod <= 3);
   CLRSCR;

   if LeewayMethod = 1 then
   begin  {LeewayMethod = Drift Rate Uncertainty}

      writeln('Refer to the Leeway Speed Formulae in the National SAR Manual');
      write('(approximately page 8-13).  Please enter the minimum ');
      writeln('leeway speed');
      writeln('at which the search object drifts (e.g., sea drogue deployed):');
      repeat  {until > 0}
         write('MINIMUM LEEWAY SPEED => ');
         readln(MinLeeSpeed)
      until MinLeeSpeed > 0;

      writeln;
      writeln('Now, enter the maximum leeway speed at which the search object');
      writeln('drifts (e.g. sea drogue NOT deployed):');
      repeat  {until > 0}
         write('MAXIMUM LEEWAY SPEED => ');
         readln(MaxLeeSpeed)
      until MaxLeeSpeed > 0;

      writeln;
      writeln('Please enter the number of hours the search object drifted:');
      repeat  {until > 0}
         write('HOURS OF DRIFT => ');
         readln(MaxHoursDrift)
      until MaxHoursDrift > 0;

      writeln;
      MaxLeeBrg := LeewayBrg;
      MinLeeDistance := MinLeeSpeed * MaxHoursDrift;
      MaxLeeDistance := MaxLeeSpeed * MaxHoursDrift

   end;  {LeewayMethod = Drift Rate Uncertainty}

   if LeewayMethod = 2 then
   begin  {LeewayMethod = Drift Time Uncertainty}

      write('Please enter the MINimum number of hours the ');
      writeln('search object drifted:');
      repeat  {until > 0}
         write('MINIMUM HOURS OF DRIFT => ');
         readln(MinHoursDrift)
      until MinHoursDrift > 0;
      writeln;
```

```
      write('Please enter the MAXimum number of hours the ');
      writeln('search object drifted:');
      repeat  {until > 0}
         write('MAXIMUM HOURS OF DRIFT => ');
         readln(MaxHoursDrift)
      until MaxHoursDrift > 0;
      writeln;

      writeln('Refer to the Leeway Speed Formulae in the National SAR Manual');
      writeln('(approximately page 8-13).  Please enter the leeway speed at');
      writeln('which the search object drifts:');
      repeat  {until > 0}
         write('MAXIMUM LEEWAY SPEED => ');
         readln(MaxLeeSpeed)
      until MaxLeeSpeed > 0;

      writelns(2);
      write('If the number of hours of search object drift ');
      writeln('is uncertain, you must');
      write('run this program twice (as mentioned ');
      writeln('earlier).  If this is the');
      write('second run, please ENTER the Leeway Speed');
      writeln('(read from the graph)');
      write('using the previously-recorded Average Surface Wind ');
      writeln('vector from the');
      writeln('shorter drift period:');
      repeat  {until valid response}
         write('MINIMUM LEEWAY SPEED => ');
         readln(MinLeeSpeed)
      until MinLeeSpeed >= 0;

      MaxLeeBrg := LeewayBrg;
      MinLeeDistance := MinLeeSpeed * MinHoursDrift;
      MaxLeeDistance := MaxLeeSpeed * MaxHoursDrift

   end;  {LeewayMethod = Drift Time Uncertainty}
   if LeewayMethod = 3 then DriftDirUncertain

end;  {LeewayDrift}

{**********************************************************************}
{Determines the sea (or slope) current affecting search object drift, and,  }
{queries user for total observed water current vector, if known.            }

procedure SeaCurrent;

begin {SeaCurrent}
   CLRSCR;
   writeln('Please enter the sea current vector as described in the National');
   writeln('SAR Manual (approximately pages 8-16i and 8-16j):');

   repeat  {until valid compass heading}
```

```
      write('SEA CURRENT DIRECTION (SET) =) ');
      readln(DirSeaCurrent)
   until (DirSeaCurrent )= 0) and (DirSeaCurrent (= 360);
   writeln;

   repeat  {until valid response}
      write('SEA CURRENT VELOCITY =) ');
      readln(SeaCurrSpeed)
   until SeaCurrSpeed )= 0;
   writeln;

   if MinHoursDrift ) 0 then
      MinSeaCurrDist := MinHoursDrift * SeaCurrSpeed;
   MaxSeaCurrDist := MaxHoursDrift * SeaCurrSpeed;

   write('Which publication did you use as the source of ');
   writeln('your sea current data?:');
   writeln;
   write('(1)  Naval Oceanographic Office Spec. Pub. ');
   writeln('Series 4000, Surface Currents');
   writeln('(2)  Publication No. 700');
   writeln('(3)  Oceanographic Atlas');
   writeln('(4)  Atlas of Surface Currents');
   writeln('(5)  Pilot Charts');
   writeln('(6)  Other');
   writeln;
   repeat  {until an above source is selected}
      write('SELECT ONLY ONE =) ');
      readln(SourceSeaCurr)
   until (SourceSeaCurr )= 1) and (SourceSeaCurr (= 6);

   CLRSCR;
   writeln('Please enter the observed total water current, if known.');
   writeln('This current is determined by observing the drift positions');
   writeln('and times of surface debris, oil slicks, or, by inserting');
   writeln('an electronic Datum Marker Buoy in the search area.  This');
   writeln('total water current vector replaces the surface wind and');
   writeln('sea current vectors previously calculated by this program.');
   writeln('(If unknown, enter a heading of 361):');
   repeat  {until valid compass heading}
      write('TOTAL WATER CURRENT DIRECTION =) ');
      readln(DirTotalCurrent)
   until (DirTotalCurrent )= 0) and (DirTotalCurrent (= 361);
   writeln;

   if DirTotalCurrent () 361 then
      repeat  {until valid response}
         write('TOTAL WATER CURRENT VELOCITY =) ');
         readln(TotCurrSpeed)
      until TotCurrSpeed ) 0;

   if MinHoursDrift ) 0 then MinTotCurrDist := MinHoursDrift * TotCurrSpeed;
```

```
   MaxTotCurrDist := MaxHoursDrift * TotCurrSpeed;

   CLRSCR;
   writeln('If you have calculated a Tidal Current vector, please enter it here');
   writeln('(If not applicable, enter a heading of 361):');
   repeat  {until valid compass heading}
      write('TIDAL CURRENT DIRECTION => ');
      readln(DirTidalCurrent)
   until (DirTidalCurrent >= 0) and (DirTidalCurrent <= 361);
   writeln;

   if DirTidalCurrent <> 361 then
   begin  {Tidal Current calculated/known}
      repeat  {until valid response}
         write('MAXIMUM TIDAL CURRENT DISTANCE => ');
         readln(MaxTideDistance)
      until MaxTideDistance >= 0;

      repeat  {until valid response}
         write('MINIMUM TIDAL CURRENT DISTANCE => ');
         readln(MinTideDistance)
      until MinTideDistance >= 0;

   end   {Tidal Current calculated/known}
end; {SeaCurrent}

{*************************************************************************}
{Determines the Datum Drift Vector (Min and Max, if applicable) from the  }
{vector sum of all previous surface drift vectors.                        }

procedure Datum;

var BrgRads,                {Individual bearings converted to radians}
    TempX,                  {Used for min Datum vector calculations}
    TempY,                  {Used for min Datum vector calculations}
    Xcomponent,             {Horizontal component of individual vectors}
    Ycomponent : real;      {Vertical component of individual vectors}

begin {Datum}

 {Start Maximum Datum Vector Calculations}
   Xcomponent := 0.0;  Ycomponent := 0.0;  {re-initialize}

   if DirTotalCurrent = 361 then  {Observed Total Water Current unknown}
   begin  {Find Wind and Sea Current vectors X & Ycomponents}
      BrgRads := DirSeaCurrent / radians;
      Xcomponent := sin(BrgRads) * MaxSeaCurrDist;
      Ycomponent := cos(BrgRads) * MaxSeaCurrDist;
      BrgRads := MaxDirWindCurrent / radians;
      Xcomponent := Xcomponent + sin(BrgRads) * MaxWindCurrDist;
      Ycomponent := Ycomponent + cos(BrgRads) * MaxWindCurrDist
   end   {Find Wind and Sea Current vectors X & Ycomponents}
```

```
    else  {Use Observed Total Water Current in lieu of Wind or Sea Currents}
    begin  {Find Observed Total Water Current vector X & Ycomponents}
       BrgRads := DirTotalCurrent / radians;
       Xcomponent := sin(BrgRads) * MaxTotCurrDist;
       Ycomponent := cos(BrgRads) * MaxTotCurrDist
    end;   {Find Observed Total Water Current vector X & Ycomponents}

    {Use the current X & Ycomponents in the Minimum Datum Vector calculations}
    {below if LeewayMethod = 1 or 3.}  TempX := Xcomponent;  TempY := Ycomponent;

    {Find Leeway Drift vector X & Ycomponents}
    BrgRads := MaxLeeBrg / radians;
    Xcomponent := Xcomponent + sin(BrgRads) * MaxLeeDistance;
    Ycomponent := Ycomponent + cos(BrgRads) * MaxLeeDistance;

    if MaxTideDistance > 0 then  {Tidal Current calculated/known}
    begin  {Find Tidal Water Current vector X & Ycomponents}
       BrgRads := DirTidalCurrent / radians;
       Xcomponent := sin(BrgRads) * MaxTideDistance;
       Ycomponent := cos(BrgRads) * MaxTideDistance
    end;   {Find Tidal Water Current vector X & Ycomponents}

    CLRSCR;
    writeln('(Max) Datum Vector:');
    AddVectors(Xcomponent,Ycomponent);
    writelns(2);
    DmaxDir := AvgWindFrom;
    DmaxDistance := ResultMagnitude;
  {End Maximum Datum Vector Calculations}

  {Start Minimum Datum Vector Calculations}
    Xcomponent := 0.0;  Ycomponent := 0.0;  {re-initialize}

    if DirTotalCurrent = 361 then  {Observed Total Water Current unknown}
    begin  {Find Min Wind and Min Sea Current vectors X & Ycomponents}
       if MinSeaCurrDist > 0 then   {If Min Sea Current vector exists}
       begin  {Find Min Sea Current vector X & Y components}
          BrgRads := DirSeaCurrent / radians;
          Xcomponent := sin(BrgRads) * MinSeaCurrDist;
          Ycomponent := cos(BrgRads) * MinSeaCurrDist
       end;  {Find Min Sea Current vector X & Y components}

       if MinDirWindCurrent <> 361 then   {If Min Wind Current vector exists}
       begin  {Find Min Wind Current vector X & Y components}
          BrgRads := MinDirWindCurrent / radians;
          Xcomponent := Xcomponent + sin(BrgRads) * MinWindCurrDist;
          Ycomponent := Ycomponent + cos(BrgRads) * MinWindCurrDist
       end    {Find Min Wind Current vector X & Y components}
    end    {Find Min Wind and Min Sea Current vectors X & Ycomponents}

    else  {Use Observed Total Water Current in lieu of Wind or Sea Currents}
```

```
    begin  {Find Observed Total Water Current vector X & Ycomponents}
       BrgRads := DirTotalCurrent / radians;
       Xcomponent := sin(BrgRads) * MinTotCurrDist;
       Ycomponent := cos(BrgRads) * MinTotCurrDist
    end;  {Find Observed Total Water Current vector X & Ycomponents}

    if LeewayMethod <> 2 then
    begin  {Set Min X & Y component values = Max Sea and WindCurrent values}
       Xcomponent := TempX;  Ycomponent := TempY
    end;  {Set Min X & Y component values = Max Sea and WindCurrent values}

    {Find Leeway Drift vector X & Ycomponents}
    BrgRads := MinLeeBrg / radians;
    Xcomponent := Xcomponent + sin(BrgRads) * MinLeeDistance;
    Ycomponent := Ycomponent + cos(BrgRads) * MinLeeDistance;

    if MinTideDistance > 0 then  {Tidal Current calculated/known}
    begin  {Find Tidal Water Current vector X & Ycomponents}
       BrgRads := DirTidalCurrent / radians;
       Xcomponent := Xcomponent + sin(BrgRads) * MinTideDistance;
       Ycomponent := Ycomponent + cos(BrgRads) * MinTideDistance
    end;  {Find Tidal Water Current vector X & Ycomponents}

    writeln('(Min) Datum Vector:');
    AddVectors(Xcomponent,Ycomponent);
    writelns(2);
    DminDir := AvgWindFrom;
    DminDistance := ResultMagnitude;
  {End Minimum Datum Vector Calculations}

    write('Total Surface Drift Dir. :   ',DminDir:3:0);
    writeln('-degrees      ',DmaxDir:3:0,'-degrees');
    write('Total Surface Drift Dist.:   ',DminDistance:4:2);
    writeln(' naut. mi.   ',DmaxDistance:4:2,' naut. mi.');
    writelns(7);
    write('HIT RETURN (Once or twice, as necessary) TO CONTINUE ');
    readln(continue);

    CLRSCR;
    writelns(4);
    writeln('Calculating . . . Please stand by')

end;  {Datum}

{*************************************************************************}
{Provides user with program information, limitations on use and license.     }

procedure Warranty;

var continue : char;  {Bogus read variable provides time to read warranty}

begin  {Warranty}
```

```
   write('(**************************************');
   writeln('**************************************)');
   write('(*          SEARCH PLANNING SOFTWARE ');
   writeln('(PROGRAM #2 OF 3)                   *)');
   write('(*  TITLE:          SURFDRIF.COM (Surface Drift ');
   writeln('Algorithm)                  *)');
   write('(*  VERSION:        1.1  for CP/M ');
   writeln('Operating System                        *)');
   write('(*  DATE WRITTEN:  September  1984');
   writeln('                                        *)');
   write('(*  LICENSE:        COPYRIGHT 1984');
   writeln('     D. RICK DOUGLAS                    *)');
   write('(**************************************');
   writeln('**************************************)');
   write('The  author  makes no express or implied ');
   writeln('warranty of any kind with  regard  to');
   write('this program material,  including, but ');
   writeln('not limited to, the implied warranty of');
   write('fitness  for  a  particular  purpose.  ');
   writeln('The  author shall  not  be  liable  for');
   write('incidental  or  consequential  damages ');
   writeln('in connection with or  arising  out  of');
   write('furnishing, use, or performance of this ');
   writeln('program.  The reader MUST HAVE a solid');
   write('understanding  of search and rescue ');
   writeln('methodology before using this software  in');
   write('making  decisions where human life is at ');
   writeln('risk.   In fact,  since no amount  of');
   write('testing  can uncover 100% of program ');
   writeln('errors,  this program is recommended  for');
   write('training  use  only.   Prior  attendance ');
   writeln('at the United  States  Coast  Guard''s');
   writeln('National SAR School is highly-encouraged.');
   writeln;
   write('(*****************************');
   writeln('  WARNING!  ***********************************)');
   write('(*      THIS  SOFTWARE  MAY  BE  FREELY-');
   writeln('DISTRIBUTED  PROVIDED  NO  FEE      *)');
   write('(*          IS  CHARGED  AND  THIS');
   writeln('  COPYRIGHT  NOTICE  IS  RETAINED        *)');
   write('(**************************************');
   writeln('**************************************)');
   writeln;
   write('PLEASE HIT RETURN (Once or twice, as necessary) TO CONTINUE');
   readln(continue)

end;  {Warranty}

{****************************************************************************}
{Called by procedure WriteToDisk; Prints out record of Wind and Sea Current  }
{vectors or Observed Total Water Current vector, as applicable.              }
```

```
procedure RecordCurrents;

var I,J        : integer;  {Used as counters}

begin  {RecordCurrents}

   if DirTotalCurrent = 361 then
   begin  {Observed Total Water Current unavailable}

      {Begin record of Wind Current Vector}
      for I := 1 to N do
      begin  {repeat for each period}
         writeln(SeaDat,' ');
         writeln(SeaDat,'WIND CURRENT PERIOD #',I:1);
         write(SeaDat,'Interval   Date-Time-Group     Wind      ');
         writeln(SeaDat,'Coefficients    Contributions');

         for J := 1 to 8 do
         begin  {repeat for each interval}
            write(SeaDat,'   ',J:1,'           ',WindsOverTime[I,J,1]:2:0);
            if WindsOverTime[I,J,2] = 0 then write(SeaDat,'0000Z      ')
            else if WindsOverTime[I,J,2] = 600 then write(SeaDat,'0600Z       ')
            else write(SeaDat,WindsOverTime[I,J,2]:4:0,'Z       ');
            write(SeaDat,WindsOverTime[I,J,3]:3:0,' / ');
            write(SeaDat,WindsOverTime[I,J,4]:2:0,'    ');
            write(SeaDat,WindCoeffs[J,1]:3:0,' / ');
            write(SeaDat,WindCoeffs[J,2]:4:3,'      ');
            write(SeaDat,WindsOverTime[I,J,5]:3:0,' / ');
            writeln(SeaDat,WindsOverTime[I,J,6]:4:3)
         end;  {repeat for each interval}
         writeln(SeaDat,' ');
         write(SeaDat,'Local Wind Current This Period            ');
         writeln(SeaDat,PeriodVectors[I,1]:3:0,' / ',PeriodVectors[I,2]:4:3);
         write(SeaDat,'Number Of Hours In This Period            ');
         writeln(SeaDat,HoursWindEffect[I]:1);
         write(SeaDat,'Wind Current Vector This Period           ');
         write(SeaDat,PeriodVectors[I,1]:3:0,' / ',PeriodVectors[I,3]:4:3);
         writeln(SeaDat,' nautical miles');
         write(SeaDat,'-----------------------------------');
         writeln(SeaDat,'-----------------------------------')
      end;  {repeat for each period}

      write(SeaDat,'Total Wind Current Direction      ');
      if MaxDirWindCurrent > 100 then
         writeln(SeaDat,MaxDirWindCurrent:3:0,'-degrees True')
      else if MaxDirWindCurrent > 10 then
         writeln(SeaDat,'0',MaxDirWindCurrent:2:0,'-degrees True')
      else writeln(SeaDat,'00',MaxDirWindCurrent:1:0,'-degrees True');

      if MinHoursDrift = 0 then
      begin  {Minimum wind current vector not computed}
```

```
            write(SeaDat,'Total Wind Current Distance      ');
            writeln(SeaDat,MaxWindCurrDist:4:2,' nautical miles')
         end   {Minimum wind current vector not computed}
         else
         begin  {max and min wind current drift distances exist}
            write(SeaDat,'Total Wind Curr. Distance:  Min = ',MinWindCurrDist:4:2);
            writeln(SeaDat,' naut.mi.    Max = ',MaxWindCurrDist:4:2,' naut. mi.')
         end;   {max and min seacurrent drift distances exist}

         write(SeaDat,'================================');
         writeln(SeaDat,'================================');
         writeln(SeaDat,' ');
         writeln(SeaDat,'SEA CURRENT VECTOR:      ');

         if SourceSeaCurr = 1 then
         begin  {Source #1 used}
            write(SeaDat,'Used Naval Oceanographic Office Spec. Pub. ');
            writeln(SeaDat,'Series 4000, Surface Currents')
         end;   {Source #1 used}
         if SourceSeaCurr = 2 then writeln(SeaDat,'Used Publication No. 700');
         if SourceSeaCurr = 3 then writeln(SeaDat,'Used Oceanographic Atlas');
         if SourceSeaCurr = 4 then
            writeln(SeaDat,'Used Atlas of Surface Currents');
         if SourceSeaCurr = 5 then writeln(SeaDat,'Used Pilot Charts');
         writeln;
         if SourceSeaCurr = 6 then
            writeln(SeaDat,'Source:________________________________');

         write(SeaDat,'Sea Current Direction (Set)       ');
         if DirSeaCurrent ) 100 then
            writeln(SeaDat,DirSeaCurrent:3:0,'-degrees True')
         else if DirSeaCurrent ) 10 then
            writeln(SeaDat,'0',DirSeaCurrent:2:0,'-degrees True')
         else writeln(SeaDat,'00',DirSeaCurrent:1:0,'-degrees True');
         write(SeaDat,'Sea Current Drift Rate             ');
         writeln(SeaDat,SeaCurrSpeed:4:2,' knots');
         if MinHoursDrift = 0 then
         begin  {No minimum vector calculations}
            write(SeaDat,'Sea Current Drift Distance:       ');
            writeln(SeaDat,MaxSeaCurrDist:4:2,' knots')
         end   {No minimum vector calculations}
         else
         begin  {max and min seacurrent drift distances exist}
            write(SeaDat,'Sea Current Drift Distance: Min = ',MinSeaCurrDist:4:2);
            writeln(SeaDat,' knots      Max = ',MaxSeaCurrDist:4:2,' knots')
         end   {max and min seacurrent drift distances exist}

      end;  {Observed Total Water Current unavailable}

      if (DirTotalCurrent () 361) and (TotCurrSpeed ) 0) then
      begin  {Total water current vector is known}
         write(SeaDat,'================================');
```

```
      writeln(SeaDat,'==================================');
      writeln(SeaDat,' ');
      writeln(SeaDat,'OBSERVED TOTAL WATER CURRENT VECTOR:      ');
      write(SeaDat,'Tot Current Direction:           ');
      if DirTotalCurrent > 100 then
        writeln(SeaDat,DirTotalCurrent:3:0,'-degrees True')
      else if DirTotalCurrent > 10 then
        writeln(SeaDat,'0',DirTotalCurrent:2:0,'-degrees True')
      else writeln(SeaDat,'00',DirTotalCurrent:1:0,'-degrees True');
      write(SeaDat,'Tot Current Drift Speed:          ',TotCurrSpeed:4:2);
      writeln(SeaDat,' knots');
      if MinHoursDrift = 0 then
      begin
        write(SeaDat,'Tot Current Drift Distance:      ');
        writeln(SeaDat,MaxTotCurrDist:4:2,' knots')
      end
      else
      begin  {max and min total current drift distances exist}
        write(SeaDat,'Tot Current Drift Distance: Min =  ',MinTotCurrDist:4:2);
        writeln(SeaDat,' knots      Max = ',MaxTotCurrDist:4:2,' knots')
      end   {max and min total current drift distances exist}

    end   {Total water current vector is known}
  end;  {RecordCurrents}

  {**************************************************************************}
  {Prints out record of search planning inputs and calculations from}
  {this program.                                                      }

  procedure WriteToDisk;

  var I           : integer;  {Used as counter}
      LastDegree,             {Last known position degrees}
      LastMinute  : real;     {Last known position minutes}

  begin  {WriteToDisk}
    assign(SeaDat,'SeaData');
    rewrite(SeaDat);

    {Begin record of last known position and time}
    LastDegree := trunc(LastLatitudeKnown);
    LastMinute := round( (LastLatitudeKnown-LastDegree) * 100);
    write(SeaDat,'Missing aerospace object/pilot(s) last known position:');
    if LastDegree < 10 then write(SeaDat,' 0',LastDegree:1:0,'-')
    else write(SeaDat,' ',LastDegree:2:0,'-');
    if LastMinute < 10 then write(SeaDat,'0',LastMinute:1:0,' ')
    else write(SeaDat,LastMinute:2:0,' ');
    if LatNS = 'N' then write(SeaDat,'North') else write(SeaDat,'South');

    LastDegree := trunc(LastLongitudeKnown);
    LastMinute := round( (LastLongitudeKnown-LastDegree) * 100);
    if LastDegree < 10 then write(SeaDat,' 0',LastDegree:1:0,'-')
```

```
else write(SeaDat,'  ',LastDegree:2:0,'-');
if LastMinute < 10 then write(SeaDat,' 0',LastMinute:1:0,' ')
else write(SeaDat,' ',LastMinute:2:0,' ');
if LongEW = 'W' then writeln(SeaDat,'West') else writeln(SeaDat,'East');

{Prints a zero in front of one-digit dates in Z DTG format}
if trunc(LastDateTime / 10000) < 10 then
writeln(SeaDat,'Time: 0',LastDateTime:5:0,'Z ',LastMonth:2,'/',LastYear:4)
else
writeln(SeaDat,'Time: ',LastDateTime:6:0,'Z ',LastMonth:2,'/',LastYear:4);
write(SeaDat,'================================');
writeln(SeaDat,'================================');

{Begin record of Average Surface (Leeway) Winds}
writeln(SeaDat,'AVERAGE SURFACE WINDS:');
writeln(SeaDat,'Date-Time-Group    Hours     Wind      Contributions');
for I := 1 to N do
begin  {repeat for each wind block}
   write(SeaDat,'    ',WindsOverTime[I,1,1]:2:0);
   if WindsOverTime[I,1,2] = 0 then write(SeaDat,'0000Z          ')
   else if WindsOverTime[I,1,2] = 600 then write(SeaDat,'0600Z          ')
   else write(SeaDat,WindsOverTime[I,1,2]:4:0,'Z          ');
   write(SeaDat,HoursWindEffect[I]:1,'     ');
   write(SeaDat,WindsOverTime[I,1,3]:3:0,' / ');
   write(SeaDat,WindsOverTime[I,1,4]:2:0,'      ');
   write(SeaDat,WindsOverTime[I,1,3]:3:0,' / ');
   writeln(SeaDat,VelocityComponent[I]:3:0)
end;  {repeat for each block}

writeln(SeaDat,' ');
write(SeaDat,'Total Average Surface Wind Direction ');
if DirSurfWind > 100 then
   write(SeaDat,DirSurfWind:3:0,'-degrees True')
else if DirSurfWind > 10 then
   write(SeaDat,'0',DirSurfWind:2:0,'-degrees True')
else write(SeaDat,'00',DirSurfWind:1:0,'-degrees True');
writeln(SeaDat,' at ',SurfWndSpeed:4:2,' knots');
write(SeaDat,'================================');
writeln(SeaDat,'================================');

writeln(SeaDat,' ');
write(SeaDat,'LEEWAY DRIFT VECTOR CALCULATIONS:      ');
if LeewayMethod = 1 then
begin {Drift rate uncertainty}
   writeln(SeaDat,'Drift Rate Uncertainty');
   writeln(SeaDat,'Hours Of Drift                    ',MaxHoursDrift:2);
   write(SeaDat,'Drift Rate                  Min = ',MinLeeSpeed:4:2);
   writeln(SeaDat,' knots     Max = ',MaxLeeSpeed:4:2,' knots');
   write(SeaDat,'Leeway Direction            ');
   writeln(SeaDat,MaxLeeBrg:3:0,'-degrees True');
   if MaxLeeBrg > 100 then
      writeln(SeaDat,MaxLeeBrg:3:0,'-degrees True')
```

```
      else if MaxLeeBrg > 10 then
         writeln(SeaDat,'0',MaxLeeBrg:2:0,'-degrees True')
      else writeln(SeaDat,'00',MaxLeeBrg:1:0,'-degrees True');
      write(SeaDat,'Leeway Distance            Min = ',MinLeeDistance:4:2);
      writeln(SeaDat,' naut.mi.  Max = ',MaxLeeDistance:4:2,' naut.mi.')
   end;  {Drift rate uncertainty}

   if LeewayMethod = 2 then
   begin  {Drift time uncertainty}
      writeln(SeaDat,'Drift Time Uncertainty');
      write(SeaDat,'Hours Of Drift             Min = ',MinHoursDrift:2);
      writeln(SeaDat,'             Max = ',MaxHoursDrift:2);
      write(SeaDat,'Drift Rate                       ',MaxLeeSpeed:4:2);
      writeln(SeaDat,' knots');
      write(SeaDat,'Leeway Direction         ');
      if MaxLeeBrg > 100 then
         writeln(SeaDat,MaxLeeBrg:3:0,'-degrees True')
      else if MaxLeeBrg > 10 then
         writeln(SeaDat,'0',MaxLeeBrg:2:0,'-degrees True')
      else writeln(SeaDat,'00',MaxLeeBrg:1:0,'-degrees True');
      write(SeaDat,'Leeway Distance            Min = ',MinLeeDistance:4:2);
      writeln(SeaDat,' naut.mi.  Max = ',MaxLeeDistance:4:2,' naut.mi.')
   end;  {Drift time uncertainty}

   if LeewayMethod = 3 then
   begin  {Directional drift uncertainty}
      writeln(SeaDat,'Directional Drift Uncertainty');
      write(SeaDat,'Maximum Expected Divergence     ');
      writeln(SeaDat,MaxDivergence:2,'-degrees');
      write(SeaDat,'Drift Rate                      ');
      writeln(SeaDat,MaxLeeSpeed:5:3,' knots');
      writeln(SeaDat,'Hours Of Drift                  ',MaxHoursDrift:2);
      write(SeaDat,'Leeway Direction         Min = ');
      if MinLeeBrg > 100 then
         write(SeaDat,MinLeeBrg:3:0,'-degrees    Max = ')
      else if MinLeeBrg > 10 then
         write(SeaDat,'0',MinLeeBrg:2:0,'-degrees    Max = ')
      else write(SeaDat,'00',MinLeeBrg:1:0,'-degrees    Max = ');
      if MaxLeeBrg > 100 then
         writeln(SeaDat,MaxLeeBrg:3:0,'-degrees True')
      else if MaxLeeBrg > 10 then
         writeln(SeaDat,'0',MaxLeeBrg:2:0,'-degrees True')
      else writeln(SeaDat,'00',MaxLeeBrg:1:0,'-degrees True');
      write(SeaDat,'Leeway Distance            Min = ',MinLeeDistance:4:2);
      writeln(SeaDat,' naut.mi.  Max = ',MaxLeeDistance:4:2,' naut.mi.')
   end;  {Directional drift uncertainty}

   write(SeaDat,'================================');
   writeln(SeaDat,'================================'),
   RecordCurrents;

   if MaxTideDistance > 0 then
```

```
    begin  {Tidal Current calculated/known}
      write(SeaDat,'Tidal Current Direction:         ');
      if DirTidalCurrent > 100 then
        writeln(SeaDat,DirTidalCurrent:3:0,'-degrees True')
      else if DirTidalCurrent > 10 then
        writeln(SeaDat,'0',DirTidalCurrent:2:0,'-degrees True')
      else writeln(SeaDat,'00',DirTidalCurrent:1:0,'-degrees True');
      if MinTideDistance > 0 then
      begin  {max and min tidal current drift distances exist}
        write(SeaDat,'Tidal Curr. Drift Distance: Min =  ');
        write(SeaDat,MinTideDistance:4:2,' naut.mi.   Max = ');
        writeln(SeaDat,MaxTideDistance:4:2,' naut.mi.')
      end   {max and min tidal current drift distances exist}
      else
      begin
        write(SeaDat,'Tidal Curr. Drift Distance:      ');
        writeln(SeaDat,MaxTotCurrDist:4:2,' naut.mi.')
      end

    end;  {Tidal Current calculated/known}

    write(SeaDat,'=================================');
    writeln(SeaDat,'=================================');
    writeln(SeaDat,' ');
    writeln(SeaDat,'TOTAL SURFACE DRIFT (Dmin & Dmax) VECTORS:');
    write(SeaDat,'Tot.Surf.Drift Direction :   ');
    if DminDir > 100 then
      write(SeaDat,DminDir:3:0,'-degrees      ')
    else if DminDir > 10 then
      write(SeaDat,'0',DminDir:2:0,'-degrees      ')
    else write(SeaDat,'00',DminDir:1:0,'-degrees      ');
    if DmaxDir > 100 then
      writeln(SeaDat,DmaxDir:3:0,'-degrees True')
    else if DmaxDir > 10 then
      writeln(SeaDat,'0',DmaxDir:2:0,'-degrees True')
    else writeln(SeaDat,'00',DmaxDir:1:0,'-degrees True');
    write(SeaDat,'Tot.Surf.Drift Distance  :   ',DminDistance:4:2);
    writeln(SeaDat,' naut. mi.   ',DmaxDistance:4:2,' naut. mi.');

    write(SeaDat,'=================================');
    writeln(SeaDat,'=================================');
    writeln(SeaDat,' ');
    close(SeaDat)
  end;  {WriteToDisk}

  {*********************************************************************}
  begin  {main program}

    {initialize program variables}
    continue := 'Q';  DatumDateTime := 0.0;  DatumMonth := 0;  DatumYear := 0;
    Days := 0;  DegreesLat := 180;  DirSeaCurrent := 361.0;
    DirSurfWind := 0.0;  DirTidalCurrent := -1.0;  DirTotalCurrent := -1.0;
```

```
DmaxDir := -1.0;  DminDir := -1.0;  DmaxDistance := 0.0;
DminDistance := 0.0;  HoursElapsed := 0;  LastDateTime := 0.0;
LastDay := 0;  LastHour := 0;  LastLatitudeKnown := 0.0;
LastLongitudeKnown := 0.0;  LastMonth := 0;  LastYear := 0;
LatNS := 'Q';  LeewayBrg := 0.0;  LeewayMethod := 4;  LongEW := 'Q';
MaxDirWindCurrent := -1.0;  MinDirWindCurrent := -1.0;
MaxDivergence := 0;  MaxHoursDrift := 0;  MinHoursDrift := 0;
MaxLeeBrg := 0.0;  MinLeeBrg := 0.0;  MaxLeeDistance := 0.0;
MinLeeDistance := 0.0;  MaxLeeSpeed := 0.0;  MinLeeSpeed := 0.0;
MaxSeaCurrDist := 0.0;  MinSeaCurrDist := 0.0;  MaxTideDistance := -1.0;
MinTideDistance := -1.0;  MaxTotCurrDist := 0.0;  MinTotCurrDist := 0.0;
MaxWindCurrDist := -1.0;  MinWindCurrDist := -1.0;  N := 0;
ResultMagnitude := 0.0;  SeaCurrSpeed := -1.0;  SourceSeaCurr := 0;
SurfWndMagnitude := 0.0;  SurfWndSpeed := 0.0;  TotCurrSpeed := 0.0;

{initialize character sets}
ValidAnswers := ['Y','y','N','n'];  No := ['N','n'];  Yes := ['Y','y'];

{initialize program arrays}
for I := 1 to 8 do for J := 1 to 2 do WindCoeffs[I,J] := 0.0;
for I := 1 to max do
begin
   HoursWindEffect[I] := 0;
   VelocityComponent[I] := 0.0;
   for J := 1 to 8 do
      for K := 1 to 6 do WindsOverTime[I,J,K] := 0.0;
   for J := 1 to 2 do PeriodVectors[I,J] := 0.0
end;

Warranty;
CLRSCR;
writeln('Given the initial surface position coordinates (latitude and');
writeln('longitude) of an object on the ocean''s surface and starting');
writeln('date/time, this program calculates an updated surface position');
writeln('(Datum point) or search area for a specified (Datum) time.  If');
writeln('the updated surface position coordinates are all ready known,');
writeln('then enter N to the next question and proceed with other recovery');
writeln('planning as outlined in the National Search and Rescue Manual.');
writelns(2);
writeln('P.S.   Run this program TWICE if the number of hours of search');
writeln('object drift is uncertain.  On the first run, enter data only');
writeln('for the SHORTER drift period (later drift start time).  After');
writeln('the program calculates the Average Surface Wind and Wind Current');
writeln('vectors, record them, ABORT the program run, and CLEAR your');
writeln('microcomputer''s memory.  Then rerun the entire program, making');
writeln('sure to enter data for the LONGER drift period (earlier drift');
writeln('start time).  Enter the vectors you recorded above when asked');
writeln('by the program.');
writelns(4);
repeat  {until valid response}
   write('Do you wish to continue with this program?  (y/n) => ');
   readln(continue);
```

```
      writeln
   until (continue in ValidAnswers);

   if (continue in Yes) then
   begin  {program run}
      continue := 'Q';  {re-initialize}
      CLRSCR;
      SurfacePosition;
      CLRSCR;

      writeln('This program includes wind current calculations to determine');
      writeln('Datum.  However, according to the National SAR Manual, wind');
      writeln('currents are usually ignored in coastal, lake, river, and');
      writeln('harbor areas due to the many variable effects from the water-');
      writeln('land interface.  This program is based on the assumption of');
      writeln('open-sea search where land masses do not interfere with the');
      writeln('action of the wind on the water or on the currents generated');
      writeln('by them.  A rule of thumb is to calculate wind currents when');
      writeln('water depths are greater than 100 feet (32 meters) and at');
      writeln('distances of 20 miles (32 kilometers) or greater from shore.');
      writeln('Wind currents are not usually used inside these limits');
      writeln('except where local knowledge makes it possible to estimate');
      writeln('one.  This is especially true where, close to shore, the ');
      writeln('water''s depth increases rapidly.');
      writelns(8);
      repeat  {until valid response}
         write('Do you wish to continue with this program?  (y/n) => ');
         readln(continue);
         writeln
      until (continue in ValidAnswers);

      if (continue in Yes) then
      begin  {Ocean Current calculations}
         CLRSCR;

         writeln('Please enter the number of hours elapsed (to the NEAREST');
         writeln('HOUR) from the last known surface position (LKP) time to');
         writeln('the desired Datum time.');
         writelns(2);
         writeln('If the number of hours of search object drift is uncertain,');
         writeln('you must run this program twice (as mentioned earlier).  On');
         writeln('the first run enter here the SHORTER number of hours of');
         writeln('search object drift.  On the second run, or if two runs are');
         writeln('not applicable, enter here the difference between the Datum');
         writeln('and LKP times as requested above:');
         repeat  {until valid HoursElapsed response}
            write('ELAPSED HOURS => ');
            readln(HoursElapsed);
            writeln
         until  HoursElapsed > 0;

         WindPeriods;
```

```
         PeriodTimes;
         InputSeaWinds;
         WindLatCoeffs;
         WindCurrent;
         AvgSurfaceWind;
         LeewayDrift;
         SeaCurrent;
         Datum;
         WriteToDisk;

         CLRSCR;
         writeln;
         writeln('A record of significant input and output data used during this');
         writeln('program run is stored in an external file named "SEADATA".');
         writeln('If you desire to keep this record permanently, please rename');
         writeln('file SEADATA before running this program again!')

      end   {Ocean Current calculations}
   end   {program run}
end. {main program}
```

Variable	Program Line Number									
AddVectors	729	849	872	934	1273	1324				
AvgSurfaceWind	914	1824								
AvgWindFrom	118	739	743	744	747	748	751	752	753	850
	873	936	1275	1326						
AvgWindTo	119	752	7.3	935						
BrgRads	830	843	844	845	866	867	868	917	927	928
	929	1228	1241	1242	1243	1244	1245	1246	1251	1252
	1253	1260	1261	1262	1266	1267	1268	1286	1287	1288
	1293	1294	1295	1301	1302	1303	1312	1313	1314	1318
	1319	1320								
continue	97	883	951	1336	1349	1397	1713	1767	1769	1771
	1773	1795	1797	1799						
count	200	203	204	207	207					
DateTime	214	225	231	235	235	242	246	246	253	
Datum	1226	1827								
DatumDateTime	120	383	384	387	1713					
DatumMonth	101	365	366	366	369	369	1713			
DatumYear	102	373	1713							
Days	103	455	456	457	458	459	460	461	462	463
	464	465	466	467	498	535	558	1714		
DaysInMonth	441	497	534	555						
DegreesLat	104	331	655	773	781	880	1714			
DirSeaCurrent	121	1145	1146	1146	1241	1286	1485	1486	1487	1488
	1489	1714								
DirSurfWind	122	936	940	1591	1592	1593	1594	1595	1715	
DirTidalCurrent	123	1203	1204	1204	1207	1266	1318	1666	1667	1668
	1669	1670	1715							

DirTotalCurrent	124	1185	1186	1186	1189	1239	1251	1282	1301	1411
	1505	1512	1513	1514	1515	1516	1715			
DmaxDir	125	1275	1331	1695	1696	1697	1698	1699	1716	
DmaxDistance	127	1276	1333	1701	1716					
DminDir	126	1326	1330	1690	1691	1692	1693	1694	1716	
DminDistance	128	1327	1332	1700	1717					
DriftDirUncertain	959	1128								
HoursElapsed	107	417	938	1717	1815	1817				
HoursRemaining	400	417	418	423	426	426	430	431	433	
HoursWindEffect	164	406	408	410	412	414	417	425	430	852
	926	1439	1582	1738						
I	105	479	543	546	546	549	549	550	550	551
	551	552	552	553	558	559	559	562	564	569
	569	570	570	572	612	615	615	617	618	619
	620	621	622	626	627	628	632	634	635	639
	639	640	640	642	652	659	661	661	661	662
	770	789	793	794	795	795	798	799	800	801
	818	818	829	835	841	841	842	842	843	844
	845	848	850	851	852	852	852	864	866	867
	868	916	924	926	926	926	927	928	929	1407
	1415	1418	1424	1425	1426	1427	1428	1429	1432	1433
	1437	1437	1439	1441	1441	1539	1576	1578	1579	1580
	1581	1582	1583	1584	1585	1586	1735	1735	1736	1738
	1739	1741	1742							
InputSeaWinds	579	1821								
J	105	479	510	513	517	517	520	520	525	527
	527	528	535	536	567	569	569	570	570	581
	586	589	590	591	592	593	594	595	596	600
	601	601	605	607	607	637	639	639	640	640
	675	708	708	829	839	841	841	841	842	842
	842	843	844	845	1407	1422	1424	1424	1425	1426
	1427	1428	1429	1430	1431	1432	1433	1735	1735	1740
	1741	1742	1742							
K	105	1741	1741							
LastDateTime	129	306	307	309	1566	1567	1569	1717		
LastDay	108	310	311	485	498	500	1718			

LastDegree	1540	1548	1549	1551	1551	1552	1557	1558	1559	1559
	1560									
LastHour	109	311	405	405	406	407	407	408	409	409
	410	411	411	412	413	414	486	487	487	489
	489	491	491	494	494	1718				
LastLatitudeKnown	130	326	327	327	330	330	331	1548	1549	1718
LastLongitudeKnown	131	345	346	346	349	349	1557	1558	1719	
LastMinute	1541	1549	1553	1553	1554	1558	1561	1561	1562	
LastMonth	110	281	282	282	285	285	497	532	532	555
	1567	1569	1719							
LastYear	111	290	449	1567	1569	1719				
LatNS	98	317	319	319	319	319	320	320	320	320
	656	777	779	783	1555	1720				
LeapYearCheck	443	450	451							
LeewayBrg	132	935	971	972	973	975	976	977	1075	1123
	1720									
LeewayDrift	1004	1025								
LeewayMethod	112	1043	1044	1044	1047	1081	1128	1306	1602	1519
	1636	1720								
lines	198	204								
LongEW	99	336	338	338	338	338	339	339	339	339
	1563	1720								
max	94	164	171	188	193	1736				
MaxDirWindCurrent	133	857	873	886	1244	1448	1449	1450	1451	1452
	1721									
MaxDivergence	113	968	969	969	971	972	973	975	976	977
	1640	1722								
MaxHoursDrift	114	992	993	996	1071	1072	1076	1077	1096	1097
	1125	1157	1196	1605	1623	1643	1722			
MaxLeeBrg	135	976	977	1075	1123	1260	1609	1610	1611	1612
	1613	1614	1627	1628	1629	1630	1631	1650	1651	1652
	1653	1654	1723							
MaxLeeDistance	137	997	1077	1125	1261	1262	1616	1633	1656	1723

MaxLeeSpeed	139 985 986 996 1064 1065 1077 1105 1106 1125 1607 1624 1642 1724
MaxSeaCurrDist	141 1157 1242 1243 1495 1500 1725
MaxTideDistance	143 1211 1212 1264 1267 1268 1663 1675 1725
MaxTotCurrDist	145 1196 1252 1253 1522 1527 1680 1726
MaxWindCurrDist	147 858 874 880 880 887 1245 1246 1457 1462 1727
MinDirWindCurrent	134 899 900 900 902 1291 1293 1721
MinHoursDrift	115 1088 1089 1124 1155 1156 1195 1195 1454 1492 1519 1622 1722
MinLeeBrg	136 972 973 1312 1645 1646 1647 1648 1649 1723
MinLeeDistance	138 996 997 1076 1124 1313 1314 1615 1632 1655 1724
MinLeeSpeed	140 1056 1057 1076 1120 1121 1124 1606 1724
MinSeaCurrDist	142 1156 1284 1287 1288 1499 1725
MinTideDistance	144 1216 1217 1316 1319 1320 1671 1674 1726
MinTotCurrDist	146 1195 1302 1303 1526 1726
MinWindCurrDist	148 905 906 1294 1295 1461 1727
MonthBefore	400 532 533 534
N	106 416 422 422 425 430 541 572 610 642 835 855 864 924 1415 1576 1727
No	155 692 1732
PeriodTimes	477 1820
PeriodVectors	193 850 851 852 852 857 858 866 867 868 1437 1437 1441 1441 1742
radians	95 736 843 866 927 1241 1244 1251 1260 1266 1286 1293 1301 1312 1318
RecordCurrents	1405 1661
ResultMagnitude	149 757 851 874 937 1276 1327 1728
SeaCurrSpeed	150 1151 1152 1156 1157 1491 1728

SeaCurrent	1136	1826								
SeaDat	159	1417	1418	1419	1420	1424	1425	1426	1427	1428
	1429	1430	1431	1432	1433	1435	1436	1437	1438	1439
	1440	1441	1442	1443	1444	1447	1449	1451	1452	1456
	1457	1461	1462	1465	1466	1467	1468	1472	1473	1475
	1476	1478	1479	1482	1484	1486	1488	1489	1490	1491
	1494	1495	1499	1500	1507	1508	1509	1510	1511	1513
	1515	1516	1517	1518	1521	1522	1526	1527	1544	1545
	1550	1551	1552	1553	1554	1555	1555	1559	1560	1561
	1562	1563	1563	1567	1569	1570	1571	1574	1575	1578
	1579	1580	1581	1582	1583	1584	1585	1586	1589	1590
	1592	1594	1595	1596	1597	1598	1600	1601	1604	1605
	1606	1607	1608	1609	1611	1613	1614	1615	1616	1621
	1622	1623	1624	1625	1626	1628	1630	1631	1632	1633
	1638	1639	1640	1641	1642	1643	1644	1646	1648	1649
	1651	1653	1654	1655	1656	1659	1660	1665	1667	1669
	1670	1673	1674	1675	1679	1680	1685	1686	1687	1688
	1689	1691	1693	1694	1696	1698	1699	1700	1701	1703
	1704	1705	1706							
SourceSeaCurr	116	1172	1173	1173	1470	1475	1476	1477	1479	1481
	1728									
SurfWndMagnitude	151	937	938	1729						
SurfWndSpeed	152	938	941	1596	1729					
SurfaceDrift	92									
SurfacePosition	265	1775								
Temp	216	235	236	246	247	267	309	310	311	387
TempAngle	731	736	739	742	743	744	746	747	748	
TempDate	217	225	226	226	256	256				
TempHour	218	236	237	237	256	257				
TempMinutes	219	247	248	257						
TempMonth	441	447	451	454						
TempX	1229	1257	1308							
TempY	1230	1257	1308							
TempYearDiv	444	449	450	450						
TotCurrSpeed	153	1192	1193	1195	1196	1505	1517	1729		

ValidAnswers	157	690	719	1732	1769	1797				
VelocityComponent	171	926	928	929	1586	1739				
VerifyDTG	214	307	384							
VerifyWinds	672	806								
Warranty	1347	1745								
WindChart	650	682	712							
WindCheck	674	679	688	690	692	717	719			
WindCoeffs	177	661	662	705	706	707	708	794	795	795
	799	800	801	818	841	842	1430	1431	1735	
WindCurrent	827	1823								
WindError	676	679	697	699	699	701	705	706	707	708
WindLatCoeffs	768	1822								
WindPeriods	398	1819								
WindsOverTime	188	485	486	488	490	492	500	501	505	506
	513	517	517	520	520	525	527	527	528	535
	536	549	549	550	550	551	551	552	552	553
	558	559	559	562	564	569	569	570	570	589
	590	591	592	593	594	595	596	600	601	601
	605	607	607	617	618	619	620	621	622	626
	627	628	632	634	635	639	639	640	640	841
	841	842	842	843	844	845	926	927	1424	1425
	1426	1427	1428	1429	1432	1433	1578	1579	1580	1581
	1583	1584	1585	1741						
writelns	198	293	298	313	351	584	709	722	775	805
	884	888	932	942	949	1008	1045	1108	1139	1175
	1198	1271	1274	1325	1334	1338	1339	1746	1754	1764
	1774	1776	1792	1801	1806	1830				
WriteToDisk	1537	1828								
Xcomponent	729	736	739	740	741	745	757	757	759	831
	837	844	844	849	863	867	867	872	918	922
	928	928	934	1231	1237	1242	1245	1245	1252	1257
	1261	1261	1267	1273	1280	1287	1294	1294	1302	1308
	1313	1313	1319	1319	1324					

Ycomponent	729	736	739	740	741	745	757	757	760	832
	837	845	845	849	863	868	868	872	919	922
	929	929	934	1232	1237	1243	1246	1246	1253	1257
	1262	1262	1268	1273	1280	1288	1295	1295	1303	1308
	1314	1314	1320	1320	1324					
Yes	156	1732	1771	1799						

Operator	Program Line Number									
arctan	736									
assign	1544									
char	99	157	674	1349						
close	1706									
cos	845	868	929	1243	1246	1253	1262	1268	1288	1295
	1303	1314	1320							
input	92									
integer	116	164	198	200	400	441	480	581	652	675
	676	770	829	916	1407	1539				
output	92									
read	708									
readln	231	242	253	281	290	306	317	326	336	345
	365	373	383	600	605	626	632	688	697	717
	794	799	883	899	905	951	968	985	992	1043
	1056	1064	1071	1088	1036	1105	1120	1145	1151	1172
	1185	1192	1203	1211	1216	1336	1397	1767	1795	1815
real	153	171	177	188	193	214	219	267	444	729
	731	832	919	1232	1541					
rewrite	1545									
round	311	406	408	410	412	414	1549	1558		
sin	844	867	928	1242	1245	1252	1261	1267	1287	1294
	1302	1313	1319							
sqrt	757									
text	159									

1625	1628	1630	1631	1633	1638	1640	1642	1643	1651
1653	1654	1656	1660	1667	1669	1670	1675	1680	1686
1687	1688	1696	1698	1699	1701	1704	1705	1747	1748
1749	1750	1751	1752	1753	1755	1756	1757	1758	1759
1760	1761	1762	1763	1768	1778	1779	1780	1781	1782
1783	1784	1785	1786	1787	1788	1789	1790	1791	1796
1803	1804	1805	1807	1808	1809	1810	1811	1812	1816
1831	1832	1833	1834	1835					

130 Variables & Operators Used   2122 Occurrences

```
(***********************************************************)
(*                                                         *)
(*          SEARCH PLANNING SOFTWARE (PROGRAM #3 OF 3)     *)
(*                                                         *)
(* TITLE:      AREA.COM (Search Area Determination Algorithm) *)
(* VERSION:       1.0  for CP/M Operating System           *)
(* DATE WRITTEN:  August  1984                             *)
(* ------------------------------------------------------- *)
(* DESCRIPTION:                                            *)
(*   - User asked for search object's last known position  *)
(*     latitude and longitude                              *)
(*   - User asked to input search object's aerospace (if   *)
(*     applicable) and surface drift vectors, previously-  *)
(*     calculated in Search Planning Software Programs #1  *)
(*     and #2                                              *)
(*   - User asked to input confidence factors, navigational *)
(*     fix and dead-reckoning errors, and search number, as *)
(*     prescribed in the National Search and Rescue Manual *)
(*   - Program calculates search area radius, square miles, *)
(*     and the latitudes and longitudes of the center and  *)
(*     four corner points.  Then it creates an "audit trail"/ *)
(*     record file of program input, significant           *)
(*     calculations, and output, named "AREADATA"          *)
(* ------------------------------------------------------- *)
(* LICENSE:  COPYRIGHT 1984     D. RICK DOUGLAS            *)
(*                                                         *)
(* The author makes no express or implied warranty of any  *)
(* kind with regard to this program material, including, but *)
(* not limited to, the implied warranty of fitness for a   *)
(* particular purpose.  The author shall not be liable for *)
(* incidental or consequential damages in connection with or *)
(* arising out of furnishing, use, or performance of this  *)
(* program.  The reader MUST HAVE a solid understanding of *)
(* search and rescue methodology before using this software *)
(* in making decisions where human life is at risk.  In fact, *)
(* since no amount of testing can uncover 100% of program  *)
(* errors, this program is recommended for training use only. *)
(* Prior attendance at the United States Coast Guard's     *)
(* National SAR School is highly-encouraged.               *)
(*                                                         *)
(*        THIS  SOFTWARE  MAY  BE  FREELY-DISTRIBUTED      *)
(*        PROVIDED  NO  FEE  IS  CHARGED  AND              *)
(*        THIS  COPYRIGHT  NOTICE  IS  RETAINED.           *)
(* ------------------------------------------------------- *)
(* LANGUAGE: PASCAL                                        *)
(* USED    : Borland International, TURBO.PAS, Version 2.0 *)
(* ------------------------------------------------------- *)
(* MODULES CALLED (Sequentially listed); (OPT) = "Optional": *)
(*                                                         *)
(* Position                                                *)
(* AeroSurfVectors                                         *)
```

```
(*  AreaSearch                                                 *)
(*  FindCoordinates                                            *)
(*  NewCoordinates (Aerospace Drift Vector Position; OPT)      *)
(*  NewCoordinates (Dmax Surface Drift Position)               *)
(*  NewCoordinates (Dmin Surface Drift Position)               *)
(*  NewCoordinates (Search Area Center Point)                  *)
(*  FindXYsToCorner                                            *)
(*  NewCoordinates (Upper Right Corner Position)               *)
(*  NewCoordinates (Lower Left Corner Position)                *)
(*  WriteToDisk                                                *)
(*                                                             *)
(***************************************************************)



program SearchArea(input,output);

const radians = 57.2957795; {Standard radian conversion factor}

var LatNS,                {Indicates whether latitude is North or South}
    LongEW      : char;   {Indicates whether longitude is East or West}

    SearchNumber : integer; {Indicates which search is being calculated}

    AeroConfidence,        {Aerospace vector drift error confidence factor}
    AeroDriftDistance,     {Total aerospace drift vector distance}
    AeroError,             {Aerospace vector drift error}
    DatumLatitude,         {Latitude of search area center point}
    DatumLongitude,        {Longitude of search area center point}
    DirAeroDrift,          {Total aerospace drift vector direction}
    DistBetween,           {Distance between computed surface drift vectors}
    DmaxDir,               {Maximum combined surface drift direction}
    DminDir,               {Minimum combined surface drift direction}
    DmaxDistance,          {Maximum combined surface drift distance}
    DminDistance,          {Minimum combined surface drift distance}
    DmaxLatitude,          {Latitude of Dmax position}
    DminLatitude,          {Latitude of Dmin position}
    DmaxLongitude,         {Longitude of Dmax position}
    DminLongitude,         {Longitude of Dmin position}
    DRerrorSearcher,       {Search craft navigational fix error}
    FixErrorSearcher,      {Search craft dead-reckoning error}
    LastLatitudeKnown,     {Last known latitude of aerospace object}
    LastLongitudeKnown,    {Last known longitude of aerospace object}
    LatAeroDrift,          {Latitude of original position + Aerospace vector}
    LatLwrLeftCorner,      {Latitude of search area's lower left corner}
    LatLwrRtCorner,        {Latitude of search area's lower right corner}
    LatUprLeftCorner,      {Latitude of search area's upper left corner}
    LatUprRtCorner,        {Latitude of search area's upper right corner}
    LongAeroDrift,         {Longitude of original position + Aerospace vector}
    LongLwrLeftCorner,     {Longitude of search area's lower left corner}
    LongLwrRtCorner,       {Longitude of search area's lower right corner}
    LongUprLeftCorner,     {Longitude of search area's upper left corner}
```

```
    LongUprRtCorner,        {Longitude of search area's upper right corner}
    MaxSurfaceError,        {Maximum surface vector drift error}
    MinSurfaceError,        {Minimum surface vector drift error}
    NewLatitude,            {Sum of previous latitude + displacement vector}
    NewLongitude,           {Sum of previous longitude + displacement vector}
    ObjectDRerror,          {Search object dead-reckoning determination error}
    ObjectFixError,         {Search object navigational fix determination error}
    PreviousDriftErrs,      {Sum of previously-computed drift errors}
    AreaOfSearch,           {Total search area in nautical miles^2}
    SearchRadius,           {TotProbableErr * SearchNumber (Safety Factor)}
    SurfaceMiniMax,         {Surface drift error minimax distance}
    SurfConfidence,         {Surface vector drift error confidence factor}
    TotalDriftError,        {Sum of aerospace and surface drift errors}
    TotObjectErr,           {Search object navigational fix + DR error}
    TotProbableErr,         {Total probable search area error}
    TotSearcherErr  : real; {Search craft navigational fix + DR error}

    AreaDat : text; {External record file of program input/output data}

{****************************************************************************}
{Writes out a specified number of blank lines; can be used to clear screen.  }

procedure writelns (lines : integer);

var count : integer;

begin
  count := 0;
  while lines > count do
  begin
    writeln;
    count := count + 1
  end
end;

{****************************************************************************}
{Queries user for last known position latitude and longitude.               }

procedure Position;

begin  {Position}

  writelns(24);
  repeat {until valid latitude}
    writeln('Was search object''s last known latitude north or south?');
    write('(Enter N or S)      Answer=> ');
    readln(LatNS);
    writeln
  until (LatNS = 'N') or (LatNS = 'n') or (LatNS = 'S') or (LatNS = 's');
  if (LatNS = 'N') or (LatNS = 'n') then LatNS := 'N' else LatNS := 'S';

  repeat {until valid latitude}
```

```
      writeln('Please enter the search object''s last known latitude ');
      writeln('(For example: 25-Degrees 45-Minutes 13-Seconds = 25.4513)');
      write('LATITUDE =) ');
      readln(LastLatitudeKnown);
      if (LastLatitudeKnown ( 0) or (LastLatitudeKnown ) 90) then
        writeln('Input latitude must be between 0-90.  Try again!');
      writeln
    until (LastLatitudeKnown )= 0) and (LastLatitudeKnown (= 90);

    repeat {until valid longitude}
      writeln('Was search object''s last known longitude east or west?');
      write('(Enter E or W)      Answer=) ');
      readln(LongEW);
      writeln
    until (LongEW = 'E') or (LongEW = 'e') or (LongEW = 'W') or (LongEW = 'w');
    if (LongEW = 'E') or (LongEW = 'e') then LongEW := 'E' else LongEW := 'W';

    repeat {until valid longitude}
      writeln('Please enter the search object''s last known longitude ');
      writeln('(For example: 160-Degrees 45-Minutes 13-Seconds = 160.4513)');
      write('LONGITUDE =) ');
      readln(LastLongitudeKnown);
      if (LastLongitudeKnown ( 0) or (LastLongitudeKnown ) 180) then
        writeln('Input longitude must be between 0-180.  Try again!');
      writeln
    until (LastLongitudeKnown )= 0) and (LastLongitudeKnown (= 180)

end;  {Position}

{**********************************************************************}
{Queries user for previously-calculated Aerospace and (Max & Min) Surface   }
{Drift vectors.                                                             }

procedure AeroSurfVectors;

var aerodrift : char;  {Indicates if aerospace drift vector used in problem}

begin  {AeroSurfVectors}

    writelns(24);
    repeat
      writeln('Did you previously calculate an aerospace drift vector ');
      write('for this problem?  (y/n): ');
      readln(aerodrift)
    until (aerodrift = 'Y') or (aerodrift = 'y') or
          (aerodrift = 'N') or (aerodrift = 'n');
    writeln;

   {Start of calculations to find Aerospace Drift Error}
    if (aerodrift = 'Y') or (aerodrift = 'y') then
    begin  {Aerospace drift vector previously-calculated}

```

```
      writeln('Please enter the previously-calculated,');
      repeat  {until valid response}
         write('TOTAL AEROSPACE DRIFT DIRECTION => ');
         readln(DirAeroDrift)
      until (DirAeroDrift >= 0) and (DirAeroDrift <= 360);
      writeln;

      writeln('Please enter the previously-calculated');
      write('TOTAL AEROSPACE DRIFT DISTANCE => ');
      readln(AeroDriftDistance);
      writeln;

      write('Refer to the "Individual Drift Error" ');
      writeln('section in the National SAR');
      write('Manual (approximately page 8-27).  ');
      writeln('Please enter the desired');
      write('Aerospace Drift Error Confidence Factor ');
      writeln('(e.g, 0.125, 0.3, etc.):');

      repeat  {until 0<AeroConfidence<1 }
         write('AEROSPACE DRIFT ERROR CONFIDENCE FACTOR => ');
         readln(AeroConfidence)
      until (AeroConfidence > 0) and (AeroConfidence < 1);
      writeln;

      AeroError := AeroDriftDistance * AeroConfidence
   end;  {Aerospace drift vector previously-calculated}
 {End of calculations to find Aerospace Drift Error}

   writelns(24);
   writeln('Please enter the sum of previous drift errors');
   writeln('(Enter a zero if not applicable):');
   write('SUM OF DRIFT ERRORS => ');
   readln(PreviousDriftErrs);
   writeln;

   writeln('Please enter the previously-calculated,');
   repeat  {until valid response}
      write('MINIMUM TOTAL SURFACE DRIFT DIRECTION => ');
      readln(DminDir)
   until (DminDir >= 0) and (DminDir <= 360);
   writeln;

   writeln('Please enter the previously-calculated,');
   write('MINIMUM TOTAL SURFACE DRIFT DISTANCE => ');
   readln(DminDistance);
   writeln;

   writeln('Please enter the previously-calculated,');
   repeat  {until valid response}
      write('MAXIMUM TOTAL SURFACE DRIFT DIRECTION => ');
      readln(DmaxDir)
```

```
    until (DmaxDir >= 0) and (DmaxDir <= 360);
    writeln;

    writeln('Please enter the previously-calculated,');
    write('MAXIMUM TOTAL SURFACE DRIFT DISTANCE => ');
    readln(DmaxDistance);
    writeln;

    writeln('Please enter the measured distance between the two, previously-');
    writeln('calculated, Total Surface Drift Distance positions:');
    write('DISTANCE BETWEEN SURFACE DRIFT POSITIONS => ');
    readln(DistBetween);
    writeln
end;  {AeroSurfVectors}

{*********************************************************************}
{When given X & Ycomponents and a reference latitude/longitude by the calling}
{procedures FindCoordinates and FindXYsToCorner, this procedure calculates   }
{the updated position's latitude/longitude.                                  }

procedure NewCoordinates(var Xcomponent, Ycomponent : real;
                             TempLatitude, TempLongitude : real);

var DecimalLatitude,    {Latitude converted from Deg-Min-Sec to decimal}
    DecimalLongitude,   {Longitude converted from Deg-Min-Sec to decimal}
    Temp,               {Interim term used in calculations}
    TempDegree,         {Interim term for position in degrees only}
    TempDenom,          {Interim term for denominator}
    TempMin,            {Interim term for position in minutes only}
    TempMinSec,         {Interim term for position minutes and seconds}
    TempSec    : real;  {Interim term for position in seconds only}

begin  {NewCoordinates}
   {initialize variables}
   Temp := 0.0;  TempDegree := 0.0;  TempDenom := 0.0;  TempMin := 0.0;
   TempMinSec := 0.0;  TempSec := 0.0;

  {Convert X & Ycomponent sign values if LatNS = N or LongEW = W}
   if LatNS = 'N' then Xcomponent := Xcomponent * -1.0;

  {Convert passed latitude to decimal}
   TempDegree := trunc(TempLatitude);
   TempMinSec := TempLatitude - TempDegree;
   DecimalLatitude := TempDegree + TempMinSec * (1 + 2/3);

  {Combine passed, decimal latitude and vector to find new, decimal latitude}
   Ycomponent := Ycomponent / 60;
   TempLatitude := DecimalLatitude + Ycomponent;

  {Convert new, decimal latitude to degrees-minutes-seconds}
   TempDegree := trunc(TempLatitude);
   TempMinSec := TempLatitude - TempDegree;
```

```
   Temp := TempMinSec * 60;
   TempMin := trunc(Temp);
   TempSec := (Temp - TempMin) * 60;
   NewLatitude := TempDegree + TempMin/100 + TempSec/10000;

  {Convert passed longitude to decimal}
   TempDegree := trunc(TempLongitude);
   TempMinSec := TempLongitude - TempDegree;
   DecimalLongitude := TempDegree + TempMinSec * (1 + 2/3);

  {Combine decimal-latitude with vector to find new, decimal longitude}
   TempDenom := cos(TempLatitude / radians) * 60;
   TempLongitude := Xcomponent / TempDenom + DecimalLongitude;

  {Convert new, decimal longitude to degrees-minutes-seconds}
   TempDegree := trunc(TempLongitude);
   TempMinSec := TempLongitude - TempDegree;
   Temp := TempMinSec * 60;
   TempMin := trunc(Temp);
   TempSec := (Temp - TempMin) * 60;
   NewLongitude := TempDegree + TempMin/100 + TempSec/10000

end;  {NewCoordinates}

{*************************************************************************}
{Finds the X & Ycomponents of the search-area-center-point-displacement-   }
{vectors created by the calling procedure, FindCoordinates.  Then, it passes }
{these components to procedure NewCoordinates for it to determine search area}
{corner point latitudes/longitudes.                                         }

procedure FindXYsToCorner(var Vector1, Vector2 : integer);

var BrgRads,           {Individual compass headings converted to radians}
    Xcomponent,        {Horizontal component of individual vectors}
    Ycomponent  : real; {Vertical component of individual vectors}

begin  {FindXYsToCorner}
   BrgRads := Vector1 / radians;
   Xcomponent := sin(BrgRads) * SearchRadius;
   Ycomponent := cos(BrgRads) * SearchRadius;
   BrgRads := Vector2 / radians;
   Xcomponent := Xcomponent + sin(BrgRads) * SearchRadius;
   Ycomponent := Ycomponent + cos(BrgRads) * SearchRadius;
   NewCoordinates(Xcomponent, Ycomponent, DatumLatitude, DatumLongitude)

end;  {FindXYsToCorner}

{*************************************************************************}
{Calculates X & Ycomponents of aerospace and surface drift vectors and sends }
{them to procedure NewCoordinates for it to determine the search area center }
{point latitude/longitude.  Next, this procedure creates two vectors (1 & 2),}
{90-degrees apart, and of length equal to the search area's radius.  It sends}
```

```
{these two vectors to procedure FindXYsToCorner for it to calculate the     }
{vectors' X & Ycomponents enroute to procedure NewCoordinates, which finds  }
{the corner point latitudes/longitudes.                                     }

procedure FindCoordinates;

var  Vector1,              {Used to find get to search area corner points}
     Vector2     : integer; {Used to find get to search area corner points}
     BrgRads,              {Individual compass headings converted to radians}
     TempX,                {Xcomponent used for interim calculations only}
     TempY,                {Ycomponent used for interim calculations only}
     Xcomponent,           {Horizontal component of individual vectors}
     Ycomponent    : real;  {Vertical component of individual vectors}

begin  {FindCoordinates}
   TempX := 0.0;  TempY := 0.0;  {initialize variables}

   if AeroDriftDistance > 0 then
   begin  {Find updated coordinates of Aerospace Drift Vector displacement}
      BrgRads := DirAeroDrift / radians;
      Xcomponent := sin(BrgRads) * AeroDriftDistance;
      Ycomponent := cos(BrgRads) * AeroDriftDistance;
      TempX := Xcomponent;      TempY := Ycomponent;  {Used below}
      NewCoordinates(Xcomponent,Ycomponent,LastLatitudeKnown,
                     LastLongitudeKnown);
      LatAeroDrift := NewLatitude;
      LongAeroDrift := NewLongitude
   end;  {Find updated coordinates of Aerospace Drift Vector displacement}

  {Find updated coordinates of Maximum Surface Drift Vector displacement}
   BrgRads := DmaxDir / radians;
   Xcomponent := TempX + sin(BrgRads) * DmaxDistance;
   Ycomponent := TempY + cos(BrgRads) * DmaxDistance;
   NewCoordinates(Xcomponent,Ycomponent,LastLatitudeKnown,LastLongitudeKnown);
   DmaxLatitude := NewLatitude;
   DmaxLongitude := NewLongitude;

  {Find updated coordinates of Minimum Surface Drift Vector displacement}
   BrgRads := DminDir / radians;
   Xcomponent := TempX + sin(BrgRads) * DminDistance;
   Ycomponent := TempY + cos(BrgRads) * DminDistance;
   NewCoordinates(Xcomponent,Ycomponent,LastLatitudeKnown,LastLongitudeKnown);
   DminLatitude := NewLatitude;
   DminLongitude := NewLongitude;

  {Find resultant X & Ycomponent of Aerospace and Surface Dmax and Dmin vectors}
   BrgRads := DmaxDir / radians;
   Xcomponent := TempX + sin(BrgRads) * DmaxDistance;
   Ycomponent := TempY + cos(BrgRads) * DmaxDistance;
   BrgRads := DminDir / radians;
   Xcomponent := Xcomponent + sin(BrgRads) * DminDistance;
   Ycomponent := Ycomponent + cos(BrgRads) * DminDistance;
```

```
  {Find search area center point latitude/longitude}
   NewCoordinates(Xcomponent,Ycomponent,LastLatitudeKnown,LastLongitudeKnown);
   DatumLatitude := NewLatitude;
   DatumLongitude := NewLongitude;

  {Find search area corner points using perpendicular vectors from center point}
   Vector1 := 360;  Vector2 := 090;  {Vectors to upper right corner}
   FindXYsToCorner(Vector1,Vector2);
   LatUprRtCorner := NewLatitude;
   LongUprRtCorner := NewLongitude;

   Vector1 := 180;  Vector2 := 270;  {Vectors to lower left corner}
   FindXYsToCorner(Vector1,Vector2);
   LatLwrLeftCorner := NewLatitude;
   LongLwrLeftCorner := NewLongitude;

   LatLwrRtCorner := LatLwrLeftCorner;  {Lower right corner coordinates}
   LongLwrRtCorner := LongUprRtCorner;

   LatUprLeftCorner := LatUprRtCorner;   {Upper left corner coordinates}
   LongUprLeftCorner := LongLwrLeftCorner

end;   {FindCoordinates}

{**********************************************************************}
{Given the search object's aerospace and/or surface drift vector(s), this    }
{procedure calculates the search area.                                       }

procedure AreaSearch;

var continue : char;  {Pauses program until user continues it}

begin {AreaSearch}

 {Start of calculations to find Surface Drift Error}
   writeln('Refer to the "Individual Drift Error" section in the National SAR');
   write('Manual (approximately page 8-27).  ');
   writeln('Please enter the desired');
   writeln('Surface Drift Error Confidence Factor (e.g, 0.125, 0.3, etc.):');

   repeat  {until 0(SurfConfidence(1 }
      write('SURFACE DRIFT ERROR CONFIDENCE FACTOR =) ');
      readln(SurfConfidence)
   until (SurfConfidence ) 0) and (SurfConfidence ( 1);
   writeln;

   MinSurfaceError := DminDistance * SurfConfidence;
   MaxSurfaceError := DmaxDistance * SurfConfidence;
   SurfaceMiniMax := (MinSurfaceError + MaxSurfaceError + DistBetween +
                      PreviousDriftErrs) / 2;
 {End of calculations to find Surface Drift Error}
```

```
TotalDriftError := AeroError + SurfaceMiniMax;

writelns(24);
writeln('Refer to the "Initial Position Error" section in the National SAR');
write('Manual (approximately page 8-29).  ');
writeln('Please enter the search object''s');
writeln('Navigational Fix Error  (e.g, 0, 5, 10, or 15 naut. mi. radius),');
writeln('and, if applicable, the Navigational Dead-Reckoning (DR) Error');
writeln('(e.g, 0, 5, 10, or 15 percent of DR distance in naut. mi.):');
write('SEARCH OBJECT NAVIGATIONAL FIX ERROR => ');
readln(ObjectFixError);
writeln;
write('SEARCH OBJECT NAVIGATIONAL DR ERROR => ');
readln(ObjectDRerror);
writeln;
TotObjectErr := ObjectFixError + ObjectDRerror;

writelns(24);
writeln('Refer to the "Search Craft Error" section in the National SAR');
write('Manual (approximately page 8-29).  ');
writeln('Please enter the search craft''s');
writeln('Navigational Fix Error  (e.g, 0, 5, 10, or 15 naut. mi. radius),');
writeln('and, if applicable, the Navigational Dead-Reckoning (DR) Error');
writeln('(e.g, 0, 5, 10, or 15 percent of DR distance in naut. mi.):');
write('SEARCH CRAFT NAVIGATIONAL FIX ERROR => ');
readln(FixErrorSearcher);
writeln;
write('SEARCH CRAFT NAVIGATIONAL DR ERROR => ');
readln(DRerrorSearcher);
writeln;
TotSearcherErr :=  FixErrorSearcher + DRerrorSearcher;

TotProbableErr := TotalDriftError * TotalDriftError;
TotProbableErr := TotProbableErr + (TotObjectErr * TotObjectErr);
TotProbableErr := TotProbableErr + (TotSearcherErr * TotSearcherErr);
TotProbableErr := sqrt(TotProbableErr);

writelns(24);
writeln('You are calculating the search area for:');
writeln;
writeln('        1  The first search');
writeln('        2  The second search');
writeln('        3  The third search');
writeln('        4  The fourth search');
writeln('        5  The fifth search');
writeln;
repeat  {until valid response}
   write('Input SEARCH NUMBER => ');
   readln(SearchNumber)
until (SearchNumber > 0) and (SearchNumber < 6);
```

```
   if SearchNumber = 1 then SearchRadius := TotProbableErr * 1.1
   else if SearchNumber = 2 then SearchRadius := TotProbableErr * 1.6
   else if SearchNumber = 3 then SearchRadius := TotProbableErr * 2.0
   else if SearchNumber = 4 then SearchRadius := TotProbableErr * 2.3
   else if SearchNumber = 5 then SearchRadius := TotProbableErr * 2.5;
   if (SearchRadius - trunc(SearchRadius) ) > 0 then
      SearchRadius := trunc(SearchRadius) + 1.0;
   AreaOfSearch := 4 * (SearchRadius * SearchRadius);

   writeln;
   writeln;
   writeln('PRELIMINARY INFORMATION:');
   writeln('------------------------');
   writeln('Search Area Radius = ',SearchRadius:5:0,' naut. mi.');
   writeln('Search Area        = ',AreaOfSearch:5:0,' sq-naut.mi.');
   writelns(5);
   write('HIT RETURN (Once or twice, as necessary) TO CONTINUE');
   readln(continue);

   writelns(24);
   writelns(4);
   writeln('Calculating . . . Please stand by')

end;  {AreaSearch}

{**********************************************************************}
{Prints out record of error adjustments to calculations of search radius and }
{area.                                                                  }

procedure WriteToDisk;

begin  {WriteToDisk}
   assign(AreaDat,'AreaData');
   rewrite(AreaDat);

   write(AreaDat,'-----------------------------------');
   writeln(AreaDat,'-----------------------------------');
   write(AreaDat,'DRIFT COORDINATES              :   ');
   writeln(AreaDat,'LATITUDE      LONGITUDE');

   write(AreaDat,'Last Known Position          :    ');
   write(AreaDat,LastLatitudeKnown:7:4,'       ');
   writeln(AreaDat,LastLongitudeKnown:7:4);

   if AeroDriftDistance > 0 then
   begin  {Aerospace vector known}
      write(AreaDat,'Aerospace Drift Position      :    ');
      write(AreaDat,LatAeroDrift:7:4,'       ');
      writeln(AreaDat,LongAeroDrift:7:4)
   end;   {Aerospace vector known}

   write(AreaDat,'Surface Dmin Position         :    ');
```

```
write(AreaDat,DminLatitude:7:4,'       ');
writeln(AreaDat,DminLongitude:7:4);
write(AreaDat,'Surface Dmax Position        :    ');
write(AreaDat,DmaxLatitude:7:4,'       ');
writeln(AreaDat,DmaxLongitude:7:4);
writeln(AreaDat,' ');

write(AreaDat,'==================================');
writeln(AreaDat,'==================================');
writeln(AreaDat,'SEARCH AREA:');
writeln(AreaDat,' ');

if AeroDriftDistance ) 0 then
begin  {Aerospace vector known}
   write(AreaDat,'Aerospace Drift Distance     :    ');
   writeln(AreaDat,AeroDriftDistance:5:2,' naut. mi.');
   write(AreaDat,'Drift Error Confidence       :    ');
   writeln(AreaDat,AeroConfidence:5:3);
   write(AreaDat,'Aerospace Drift Error        :    ');
   writeln(AreaDat,AeroError:5:2,' naut. mi.');
   write(AreaDat,'----------------------------------');
   writeln(AreaDat,'----------------------------------')
end;   {Aerospace vector known}

write(AreaDat,'Sum of Previous Drift Errors  :    ');
writeln(AreaDat,PreviousDriftErrs:5:2,' naut. mi.');
write(AreaDat,'Surface Drift Distance       :    ',DminDistance:5:2);
writeln(AreaDat,' naut. mi.   ',DmaxDistance:5:2,' naut. mi.');
writeln(AreaDat,'Drift Error Confidence       :    ',SurfConfidence:5:3);
write(AreaDat,'Drift Error Min and Max      :    ',MinSurfaceError:5:2);
writeln(AreaDat,' naut. mi.   ',MaxSurfaceError:5:2,' naut. mi.');
write(AreaDat,'Distance Between Dmin/max     :    ');
writeln(AreaDat,DistBetween:5:2,' naut. mi.');
write(AreaDat,'Surface Drift Error Minimax   :    ');
writeln(AreaDat,SurfaceMiniMax:5:2,' naut. mi.');
write(AreaDat,'----------------------------------');
writeln(AreaDat,'----------------------------------');

write(AreaDat,'Total Drift Error             :    ');
writeln(AreaDat,TotalDriftError:5:2,' naut. mi.');
write(AreaDat,'----------------------------------');
writeln(AreaDat,'----------------------------------');

write(AreaDat,'Object Navigation Fix Error   : ');
writeln(AreaDat,ObjectFixError:5:0,' naut. mi.');
write(AreaDat,'Object Dead-Reckoning Error   :    ');
writeln(AreaDat,ObjectDRerror:5:2,' naut. mi.');
write(AreaDat,'Object Total Position Error   :    ');
writeln(AreaDat,TotObjectErr:5:2,' naut. mi.');
write(AreaDat,'----------------------------------');
writeln(AreaDat,'----------------------------------');
```

```
write(AreaDat,'Searcher Navigation Fix Error :  ');
writeln(AreaDat,FixErrorSearcher:5:0,' naut. mi.');
write(AreaDat,'Searcher Dead-Reckoning Error :    ');
writeln(AreaDat,DRerrorSearcher:5:2,' naut. mi.');
write(AreaDat,'Searcher Total Position Error :    ');
writeln(AreaDat,TotSearcherErr:5:2,' naut. mi.');
write(AreaDat,'-----------------------------------');
writeln(AreaDat,'-----------------------------------');

write(AreaDat,'Total Probable Error          :    ');
writeln(AreaDat,TotProbableErr:5:2,' naut. mi.');

if SearchNumber = 1 then
   writeln(AreaDat,'Safety Factor                 :     1.1')
else if SearchNumber = 2 then
   writeln(AreaDat,'Safety Factor                 :     1.6')
else if SearchNumber = 3 then
   writeln(AreaDat,'Safety Factor                 :     2.0')
else if SearchNumber = 4 then
   writeln(AreaDat,'Safety Factor                 :     2.3')
else if SearchNumber = 5 then
   writeln(AreaDat,'Safety Factor                 :     2.5');

write(AreaDat,'-----------------------------------');
writeln(AreaDat,'-----------------------------------');
write(AreaDat,'Search Radius                 :  ');
writeln(AreaDat,SearchRadius:5:0,' naut. mi.');
write(AreaDat,'Search Area                   :  ');
writeln(AreaDat,AreaOfSearch:5:0,' naut. mi.-squared.');

write(AreaDat,'-----------------------------------');
writeln(AreaDat,'-----------------------------------');
write(AreaDat,'SEARCH AREA COORDINATES:      :    ');
writeln(AreaDat,'LATITUDE      LONGITUDE');

write(AreaDat,'Center Point                  :    ');
write(AreaDat,DatumLatitude:7:4,'      ');
writeln(AreaDat,DatumLongitude:7:4);
writeln(AreaDat,' ');

write(AreaDat,'Corner Point (Upper Left)     :    ');
write(AreaDat,LatUprLeftCorner:7:4,'      ');
writeln(AreaDat,LongUprLeftCorner:7:4);

write(AreaDat,'Corner Point (Lower Left)     :    ');
write(AreaDat,LatLwrLeftCorner:7:4,'      ');
writeln(AreaDat,LongLwrLeftCorner:7:4);

write(AreaDat,'Corner Point (Upper Right)    :    ');
write(AreaDat,LatUprRtCorner:7:4,'      ');
writeln(AreaDat,LongUprRtCorner:7:4);
```

```
   write(AreaDat,'Corner Point (Lower Right)    :     ');
   write(AreaDat,LatLwrRtCorner:7:4,'       ');
   writeln(AreaDat,LongLwrRtCorner:7:4);

   write(AreaDat,'=================================');
   writeln(AreaDat,'=================================');

   close(AreaDat)
end;  {WriteToDisk}

{************************************************************************}
{Provides user with program information, limitations on use and license.     }

procedure Warranty;

var continue : char;  {Bogus read variable provides time to read warranty}

begin  {Warranty}

   write('(***********************************');
   writeln('***********************************)');
   write('(*          SEARCH PLANNING SOFTWARE ');
   writeln('(PROGRAM #3 OF 3)                   *)');
   write('(*  TITLE:         AREA.COM (Search Area ');
   writeln('Determination Algorithm)         *)');
   write('(*  VERSION:       1.0  for CP/M ');
   writeln('Operating System                        *)');
   write('(*  DATE WRITTEN:  August  1984');
   writeln('                                          *)');
   write('(*  LICENSE:       COPYRIGHT 1984');
   writeln('     D. RICK DOUGLAS                      *)');
   write('(***********************************');
   writeln('***********************************)');
   write('The  author  makes no express or implied ');
   writeln('warranty of any kind with  regard  to');
   write('this program material,  including, but ');
   writeln('not limited to, the implied warranty of');
   write('fitness  for  a  particular  purpose.  ');
   writeln('The  author shall  not  be  liable  for');
   write('incidental  or  consequential  damages ');
   writeln('in connection with or  arising  out  of');
   write('furnishing, use, or performance of this ');
   writeln('program.  The reader MUST HAVE a solid');
   write('understanding  of search and rescue ');
   writeln('methodology before using this software  in');
   write('making  decisions where human life is at ');
   writeln('risk.   In fact,  since no amount  of');
   write('testing  can uncover 100% of program ');
   writeln('errors,  this program is recommended  for');
   write('training  use  only.   Prior  attendance ');
   writeln('at the United  States  Coast  Guard''s');
   writeln('National SAR School is highly-encouraged.');
```

```
   writeln;
   write('(****************************');
   writeln('  WARNING!  ***********************************)');
   write('(*      THIS  SOFTWARE  MAY  BE  FREELY-');
   writeln('DISTRIBUTED  PROVIDED  NO  FEE      *)');
   write('(*         IS  CHARGED  AND  THIS');
   writeln('   COPYRIGHT  NOTICE  IS  RETAINED         *)');
   write('(************************************');
   writeln('*************************************)');
   writeln;
   write('PLEASE HIT RETURN (Once or twice, as necessary) TO CONTINUE');
   readln(continue)

end;   {Warranty}

{**********************************************************************}

begin  {Main Program}

   {Initialize program variables}
   AeroConfidence := 0.0;  AeroDriftDistance := 0.0;  AeroError := 0.0;
   DatumLatitude := 0.0;  DatumLongitude := 0.0;  DirAeroDrift := 0.0;
   DistBetween := 0.0;  DmaxDir := 0.0;  DmaxDir := 0.0;
   DmaxDistance := 0.0;  DminDistance := 0.0;  DmaxLatitude := 0.0;
   DmaxLongitude := 0.0;  DminLatitude := 0.0;  DminLongitude := 0.0;
   DRerrorSearcher := 0.0;  FixErrorSearcher := 0.0;  LastLatitudeKnown := 0.0;
   LastLongitudeKnown := 0.0;  LatAeroDrift := 0.0;  LatLwrLeftCorner := 0.0;
   LatLwrRtCorner := 0.0;  LatNS := 'Q';  LatUprLeftCorner := 0.0;
   LatUprRtCorner := 0.0;  LongAeroDrift := 0.0;
   LongEW := 'Q';  LongLwrLeftCorner := 0.0;  LongLwrRtCorner := 0.0;
   LongUprLeftCorner := 0.0;  LongUprRtCorner := 0.0;  MaxSurfaceError := 0.0;
   MinSurfaceError := 0.0;  NewLatitude := 0.0;  NewLongitude := 0.0;
   ObjectDRerror := 0.0;  ObjectFixError := 0.0;  PreviousDriftErrs := 0.0;
   AreaOfSearch := 0.0;  SearchNumber := 0;  SearchRadius := 0.0;
   SurfaceMiniMax := 0.0;  SurfConfidence := 0.0;  TotalDriftError := 0.0;
   TotObjectErr := 0.0;  TotProbableErr := 0.0;  TotSearcherErr := 0.0;

   Warranty;
   Position;
   AeroSurfVectors;
   AreaSearch;
   FindCoordinates;
   WriteToDisk;
   writelns(24);
   writeln('A record of significant input and output data used during this');
   writeln('program run is stored in an external file named "AREADATA".');
   writeln('If you desire to keep this record permanently, please rename');
   writeln('file AREADATA before running this program again!')

end.  {Main Program}
```

Variable	Program Line Number									
AeroConfidence	76	229	230	230	233	589	748			
aerodrift	191	199	200	200	201	201	205	205		
AeroDriftDistance	77	217	233	381	384	385	564	584	587	748
AeroError	78	233	469	591	748					
AeroSurfVectors	189	767								
AreaDat	121	552	553	555	556	557	558	560	561	562
	566	567	568	571	572	573	574	575	576	577
	579	580	581	582	586	587	588	589	590	591
	592	593	596	597	598	599	600	601	602	603
	604	605	606	607	608	610	611	612	613	615
	616	617	618	619	620	621	622	624	625	626
	627	628	629	630	631	633	634	637	639	641
	643	645	647	648	649	650	651	652	654	655
	656	657	659	660	661	662	664	665	666	668
	669	670	672	673	674	676	677	678	680	681
	683									
AreaOfSearch	112	527	534	652	761					
AreaSearch	445	768								
BrgRads	344	349	350	351	352	353	354	372	383	384
	385	394	395	396	402	403	404	410	411	412
	413	414	415							
continue	447	537	691	739						
count	128	131	132	135	135					
DRerrorSearcher	91	497	499	627	753					
DatumLatitude	79	355	419	660	749					
DatumLongitude	80	355	420	661	749					
DecimalLatitude	283	303	307							
DecimalLongitude	284	320	324							
DirAeroDrift	81	211	212	212	383	749				

DistBetween	82	271	465	604	750					
DmaxDir	83	259	260	260	394	410	750	750		
DmaxDistance	85	265	395	396	411	412	464	599	751	
DmaxLatitude	87	398	575	751						
DmaxLongitude	89	399	576	752						
DminDir	84	247	248	248	402	413				
DminDistance	86	253	403	404	414	415	463	598	751	
DminLatitude	88	406	572	752						
DminLongitude	90	407	573	752						
FindCoordinates	368	769								
FindXYsToCorner	342	424	429							
FixErrorSearcher	92	494	499	625	753					
LastLatitudeKnown	93	159	160	160	163	163	387	397	405	418
	561	753								
LastLongitudeKnown	94	177	178	178	181	181	388	397	405	418
	562	754								
LatAeroDrift	95	389	567	754						
LatLwrLeftCorner	96	430	433	669	754					
LatLwrRtCorner	97	433	677	755						
LatNS	71	150	152	152	152	152	153	153	153	153
	298	755								
LatUprLeftCorner	98	436	665	755						
LatUprRtCorner	99	425	436	673	756					
lines	126	132								
LongAeroDrift	100	390	568	756						
LongEW	72	168	170	170	170	170	171	171	171	171
	757									
LongLwrLeftCorner	101	431	437	670	757					

LongLwrRtCorner	102	434	678	757						
LongUprLeftCorner	103	437	666	758						
LongUprRtCorner	104	426	434	674	758					
MaxSurfaceError	105	464	465	602	758					
MinSurfaceError	106	463	465	601	759					
NewCoordinates	280	355	387	397	405	418				
NewLatitude	107	315	389	398	406	419	425	430	759	
NewLongitude	108	332	390	399	407	420	426	431	759	
ObjectDRerror	109	482	484	618	760					
ObjectFixError	110	479	484	616	760					
Position	142	766								
PreviousDriftErrs	111	241	466	597	760					
radians	69	323	349	352	383	394	402	410	413	
SearchArea	67									
SearchNumber	74	517	518	518	520	521	522	523	524	636
	638	640	642	644	761					
SearchRadius	113	350	351	353	354	520	521	522	523	524
	525	525	526	526	527	527	533	650	761	
SurfConfidence	115	459	460	460	463	464	600	762		
SurfaceMiniMax	114	465	469	606	762					
Temp	285	294	312	313	314	329	330	331		
TempDegree	286	294	301	302	303	310	311	315	318	319
	320	327	328	332						
TempDenom	287	294	323	324						
TempLatitude	281	301	302	307	310	311	323			
TempLongitude	281	318	319	324	327	328				
TempMin	288	294	313	314	315	330	331	332		
TempMinSec	289	295	302	303	311	312	319	320	328	329

TempSec	290	295	314	315	331	332				
TempX	373	379	386	395	403	411				
TempY	374	379	386	396	404	412				
TotObjectErr	117	484	502	502	620	763				
TotProbableErr	118	501	502	502	503	503	504	504	520	521
	522	523	524	634	763					
TotSearcherErr	119	499	503	503	629	763				
TotalDriftError	116	469	501	501	611	762				
Vector1	342	349	370	423	424	428	429			
Vector2	342	352	371	423	424	428	429			
Warranty	689	765								
writelns	126	146	195	237	471	486	506	535	539	540
	771									
WriteToDisk	549	770								
Xcomponent	280	298	298	324	345	350	353	353	355	375
	384	386	387	395	397	403	405	411	414	414
	418									
Ycomponent	280	306	306	307	346	351	354	354	355	376
	385	386	387	396	397	404	405	412	415	415
	418									

<u>Operator</u>	<u>Program Line Number</u>							
assign	552							
char	72	191	447	691				
close	683							
cos	323	351	354	385	396	404	412	415
input	67							
integer	74	126	128	342	371			
output	67							

readln	150	159	168	177	199	211	217	229	241	247
	253	259	265	271	459	479	482	494	497	517
	537	739								
real	119	280	281	290	346	376				
rewrite	553									
sin	350	353	384	395	403	411	414			
sqrt	504									
text	121									
trunc	301	310	313	318	327	330	525	526		
write	149	158	167	176	198	210	216	220	222	224
	228	240	246	252	258	264	270	453	458	473
	478	481	488	493	496	516	536	555	557	560
	561	566	567	571	572	574	575	579	586	588
	590	592	596	598	601	603	605	607	610	612
	615	617	619	621	624	626	628	630	633	647
	649	651	654	656	659	660	664	665	668	669
	672	673	676	677	680	695	697	699	701	703
	705	707	709	711	713	715	717	719	721	723
	725	729	731	733	735	738				
writeln	134	148	151	156	157	161	162	166	169	174
	175	179	180	197	202	208	213	215	218	221
	223	225	231	238	239	242	244	249	251	254
	256	261	263	266	268	269	272	452	454	455
	461	472	474	475	476	477	480	483	487	489
	490	491	492	495	498	507	508	509	510	511
	512	513	514	529	530	531	532	533	534	541
	556	558	562	568	573	576	577	580	581	582
	587	589	591	593	597	599	600	602	604	606
	608	611	613	616	618	620	622	625	627	629
	631	634	637	639	641	643	645	648	650	652
	655	657	661	662	666	670	674	678	681	696
	698	700	702	704	706	708	710	712	714	716
	718	720	722	724	726	727	728	730	732	734
	736	737	772	773	774	775				

96 Variables & Operators Used   951 Occurrences

Vita

Don Richard "Rick" Douglas was born in Salt Lake City, Utah on August 4, 1953. After graduating in 1975 from the United States Air Force Academy with a Bachelor of Science Degree in history, he attended helicopter pilot training in Alabama. He served as a Combat Rescue Aircraft Commander in USAF HH-53 "Super Jolly Green Giants" in Japan (air-sea rescue) from 1976-79 and USAF UH-1N "Twin Hueys" in Utah (mountain & desert rescue) from 1979-81. He was then "loaned" to the United States Coast Guard (USAF Exchange Officer Program), where he again flew air-sea rescue missions in USCG HH-3F "Pelicans" in Florida from 1981-83. He is credited with saving 20 lives and assisting another 26 to safety during 30 separate air search and rescue missions.

Rick is a graduate of the following rescue-related training courses: USAF Basic Survival in Colorado, USAF Jungle Survival in the Philippines, USN Aircraft Underwater Egress in Florida, USN SCUBA and Drown-Proofing in California, USA and West German Airborne, and USCG National Search and Rescue School in New York.

He is married to a criminologist and ex-New Orleans Playboy Bunny, the former Paula J. Hartzfeld of Bronaugh, Missouri. Their son, Craig Madison, is also credited with a "save". Craig received numerous newspaper and Scouting awards for saving the life of a Utah man who was seriously-injured in a highway accident in 1979.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release; distribution unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
AFIT/GSO/MATH/84D-1

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
School of Engineering

6b. OFFICE SYMBOL *(If applicable)*
AFIT/EN

7a. NAME OF MONITORING ORGANIZATION

6c. ADDRESS *(City, State and ZIP Code)*
Air Force Institute of Technology
Wright-Patterson AFB, Ohio 45433

7b. ADDRESS *(City, State and ZIP Code)*

8a. NAME OF FUNDING/SPONSORING ORGANIZATION

8b. OFFICE SYMBOL *(If applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State and ZIP Code)*

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.

11. TITLE *(Include Security Classification)*
See Box #19 (Unclassified)

12. PERSONAL AUTHOR(S)
Douglas, Don Richard

13a. TYPE OF REPORT
Thesis

13b. TIME COVERED
FROM ______ TO ______

14. DATE OF REPORT *(Yr., Mo., Day)*
1984 December 14

15. PAGE COUNT
186

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.
08	03	
06	07	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Coast Guard, Computer Applications, Microcomputer Pascal, Search and Rescue.

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Title: Microcomputer Application of Aerospace Asset Surface Search Planning

Approved for public release: IAW AFR 190-17.
LYNN E. WOLAVER
Dean for Research and Professional Development

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL
DR. PANNA B. NAGARSENKER

22b. TELEPHONE NUMBER *(Include Area Code)*
513-255-7210

22c. OFFICE SYMBOL
AFIT/ENC

DD FORM 1473, 83 APR   EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

This paper tackles the most time-consuming and complicated type of search and recovery planning -- calculating the approximate surface position of an aerospace object which has been affected over time by glide or parachute winds aloft, as well as surface current winds, leeway drift, and sea current vectors. The three, highly-interactive, search applications programs herein are written in Standard Pascal using Borland International's "TURBO Pascal" (an inexpensive software package available for virtually every microcomputer on the market). They have been tested on a small, portable, 64K memory, Z-80A processor-based microcomputer (Osborne One), a Convergent Technologies C-3 Data System, and a Digital Equipment Corporation VAX 11-780 mainframe.

Search and Rescue/Recovery (SAR) in the United States is based on the humanitarian principle which compels people to render aid to those in distress. Search planning guidelines and formulae to help locate persons in distress or missing aerospace objects are described in the National Search and Rescue Manual (AFM 64-2). This methodo~~logy~~ has not been implemented for microcomputers in a compiled, transportable programming language like Pascal. This research project does just that. It does not, however, teach the guidelines or formulae. The reader MUST have a solid understanding of SAR methodo~~logy~~ before using ~~the~~ attached software package to assist in making decisions where human life is at risk. In fact, since no amount of testing can uncover 100% of program errors, the attached software package is recommended for training use only.

Appendices include: glossary, user's guide, sample problem with program runs and solutions, and source code listing.

END

FILMED

4-85

DTIC